HCJournal™
Home Computing Journal

Welcome to: Volume 2

—Intelligent Software for Personal Computers in a Home Environment

PRODUCTION NOTE: This printed journal was produced in camera-ready format with a desktop publishing system employing an Apple Macintosh and LaserWriter system, and the following software: MacPaint, MacDraw, and MacWrite from Apple Computer, Inc; PageMaker from Aldus Corporation; ReadySetGo from Manhattan Graphics; plus Desktop Magic and LaserMagic from World Class Software.

Creative Staff:
Randy Thompson
Roger Wood
Norman Winney

Editorial Consultant
Gary M. Kaplan

HCJ FEATURES

HCJ TECH NOTES

HCJ FOCUS

HCJ DIRECTOR

Home Computing Journal (HCJ) is a quarterly multi-media software subscription service containing ready-to-run productivity, education, entertainment, and utlity programs on a floppy disk. The accompanying workbook contains the required support documentation plus additional technical notes and programming aids. For current single-copy and subscription pricing, please see last page. HCJ assumes no liability for errors in articles, programs, or workbook material.

Individual volumes, bulk orders, and subscriptions are available.
See last page and /or inquire for current pricing and catalog:
Home Computing Journal
P.O. Box 70248 • Eugene, OR 97401
Tel. (503) 342-4013

FormFlex

No more writer's cramps... no more loose papers cluttering your desktop. With this powerful-yet-flexible utility, you'll put everything in perfect form.

Reports, expense records, data sheets, contracts, purchase receipts... In today's complex society, it seems like the most common means of communicating and organizing our transactions, thoughts, records, and activities are *forms*. All-purpose forms are not always suited to our needs, however, and we often find ourselves altering them to fit our requirements. But rather than scratching out words or scribbling in text between the lines, why not create your own custom forms? *FormFlex* lets you do just that.

With *FormFlex*, you can design forms and print them with a standard 80-column printer. These forms have an advantage over pre-made forms in more ways than one: You can store the forms on a disk to edit and print as the need arises; you can create special data fields; and you can actually perform calculations in these fields, using addition, subtraction, multiplication, and division.

```
                        ABC Company
                    Cost Comparison Form
--------------------------------------------------------------------------------
     Project No.: _______                                  Date: _______
    Company Name: ____________________
         Contact: ____________________                     Tel: _____________
         Address: ____________________
            City: ____________________
           State: __  Zip: _____
--------------------------------------------------------------------------------
                        Item Descriptions
Item A: ____________________
   A1 Source: ____________________
   A2 Source: ____________________
   A3 Source: ____________________
   A4 Source: ____________________
Item B: ____________________
   B1 Source: ____________________
   B2 Source: ____________________
   B3 Source: ____________________
   B4 Source: ____________________
--------------------------------------------------------------------------------
    Low                                          High      Average    High-Low
   Quote 1      Quote 2      Quote 3            Quote 4     Quote     % Diff.
--------------------------------------------------------------------------------

A1 $_______  A2 $_______  A3 $_______  A4 $_______  Aav$_______  Adf ____%

B1 $_______  B2 $_______  B3 $_______  B4 $_______  Bav$_______  Bdf ____%

--------------------------------------------------------------------------------

High A $_______   High B $_______   High Total $_________

 Low A $_______    Low B $_______    Low Total $_________

                                     Average: _______

                                  % Difference: ____%
```

Figure 1.
This is the sample form that you will find on your HCJ Volume 2 disk with the file name: COSTCOMP.

Getting Started

Before you can use *FormFlex*, you must first create a work disk. This is done by copying certain files from your HCJ Volume 2 disk to a notched (non-write-protected) disk. See your computer's accompanying sidebar for specific instructions on creating a *FormFlex* work disk for your computer.

In Good Form

When you **RUN** *FormFlex*, the program displays a title screen during initialization, then takes you to the main menu screen:

```
FormFlex

1) Make Form
2) Use Form
3) Exit FormFlex
```

The first two options actually represent two separate programs that are loaded from disk when selected. For this reason, it is important that you keep your work disk in the drive at all times.

Using *FormFlex* requires two steps: The first step is to *make* a form (option 1). This master form comprises the standard information that will appear on the final printout, and a list of all the different information fields. You can save each form to a disk and call it up later for re-use or modification. The second step is to *use* the form (option 2) by entering data into each field of the master form, and printing the finished product. You may also save the field information to disk for later editing or for merging into other forms.

To help you learn how to use *FormFlex*, we are going to show you a simple form that we have created, and how it can be customized for your own purposes. We have provided this form on your HCJ Volume 2 disk under the file name COSTCOMP. Note that an extension of 2 to 4 characters is used by the program, and is not entered by the user when Loading or Saving from the program. For a list of these extensions, see your computer's "Special Hints" sidebar. We suggest that you work directly with the program while reading these instructions. For starters, create a work disk, load and run *FormFlex*, and then select the Make Form option from the above menu.

Make Form

When selected, the Make Form program module is loaded from disk, and the following menu is presented:

```
Make Form

1) Edit Form
2) Erase Form
3) Change Formula or Edit Field
4) Disk Routines
5) Printer Routines
6) Return to FormFlex Menu
```

First, let's load our sample form into memory. This is accomplished via the Disk Routines menu. To get this menu, select option 4 from the above menu. This is what you should see:

```
Disk Routines

1) Load Form
2) Save Form
3) Return to Make Form Menu
```

By choosing option 1, we can Load our form. When the computer prompts you, enter the file name COSTCOMP. As you can see, if we were to create a new form or modify an old one, we could Save the form using option 2 of the Disk Routines menu. Whenever you Save a form, the computer checks the disk to see if a form with the same name already exists. If it does, you are asked if you wish to replace the existing form with the current one.

Once our file is loaded, choose option 3 to return to the Make Form menu. Now, let's take a look at our form. To do this, select option 1, Edit Form.

Edit Form

This part of the program is where you create new forms and edit previously existing ones. Here, you are provided with a simple text editor for the designing of forms. Before we make any changes to the form that we have just loaded, let's go through a brief explanation of the Editor.

The Editor's screen layout is as follows: The top few lines of the screen are for displaying messages and receiving information such as file names. The rest of the screen is for the actual editing of a form. The upper-left corner of the screen shows the current row and column position of the cursor. The row can vary between 1 and 60 and the column between 1 and 80. These dimensions cover an 8-1/2" by 11" piece of paper. The computer's screen acts as a window into this page of text. You can move your window to display any part of the page that you desire. When you move your cursor off the screen, the computer automatically updates its viewing window in order to show the section of the page that the cursor has moved to.

All of your computer's normal editing functions are available. There are even some extra ones such as deleting and inserting whole lines of text, and paging quickly through the form. For a list of editing keys and their functions, refer to your computer's Control Capsule.

Text & Fields

Any text you enter during Make Form will appear on the final printed form just as you enter it. On the other hand, if you wish to designate a "blank" on the form to be filled in with certain information, you must declare it as a "field." There are 5 different types of fields that you can specify on any form: Alpha, Uppercase, Edit Characters, Numeric, and Formula. These fields are represented on the screen by the inverse letters a for Alpha, u for Uppercase, e for Edit Characters, n for Numeric, and f for Formula. Your computer's Control Capsule shows you the exact keystrokes required to enter these characters. The only other inverse character is one designating a decimal-point location in either a numeric or formula field. By selecting a decimal location, you determine the format any number will have in a filled-out form. Again, your Control Capsule shows you how to enter this character.

Our Sample File—Cost Comparison Form

To best understand the rules for using *FormFlex*, we will refer to our sample form, COSTCOMP (see Figure 1). Note that this sample is only provided to demonstrate the different functions available in *FormFlex*; it should *not* be viewed as all this powerful "form processor" is capable of producing. COSTCOMP is designed for a hypothetical company (the ABC Company) to compare the cost of various bids for parts from four sources. By using the *FormFlex* Formula feature, the program provides calculation of the average bid for each of two items (designated A and B), and the percent difference between the highest and lowest bid. It also calculates the highest and lowest total bid for both items, along with the average total bid and the percent difference between high and low bids.

Across the top, is the header for the form—ABC Company, etc. This is not a field or a field name—it's just text entered from the Make Form Editor. A field name comprises the characters immediately to the left of any field of inverse characters. In this case, the first field name is "Project No.:"—with the 8 inverse n characters on the screen (depicted

Figure 2. Field Types

Name	You see	Legal Characters
Alpha	a	All printable characters
Uppercase	u	Uppercase letters only
Edit Characters	e	You choose the legal characters
Numeric	n	Digits, decimal point, and minus sign
Formula	f	Contents of field is calculated by formula

Maximum Lengths of Fields and Field Names

Computer	Maximum Field Length	Maximum Field Name Length
Apple	79	79
Atari	25	15
C-64	25	15
IBM	79	79
TI-99/4A	25	15

Atari, C-64, & TI-99/4A Users: A field name may actually exceed 15 characters, but only the rightmost 15 characters are considered to be the field name. In other words, if you declare a field name to be

```
Cost per thousand
```

the computer will only consider the rightmost 15 characters as the actual field name in formulas, etc.—i.e.,

```
st per thousand
```

Apple & IBM Users: All fields must be on the same line. Therefore, both the Apple and IBM programs allow a combined total-length maximum of 80 characters for Field Name Length and Field Length (i.e., Field Name Length + Field Length must be less than or equal to 80).

SPECIAL HINTS

Special Note: *FormFlex* requires either an Apple IIe with an 80-column card or an Apple IIc.

Making an Apple *FormFlex* Work Disk

Format a ProDOS disk using the HCJ Format program supplied on your HCJ Volume 2 disk. Return to Applesoft BASIC, then **LOAD** each of the following programs into memory, and, in turn, **SAVE** each file to the newly formatted disk.

```
FORMFLEX
MAKEFORM
USEFORM
```

To use the demonstration file COSTCOMP, follow these steps:

1. With your *FormFlex* work disk in the startup drive, type
   RUN FORMFLEX
2. Select option 1, Make Form.
3. Select option 4, Disk Routines.
4. Place the HCJ Volume 2 disk in the startup drive.
5. Select option 1, Load Form.
6. Enter the file name COSTCOMP
7. After the file has finished loading, remove the HCJ Volume 2 disk from the drive and put your *FormFlex* work disk in the drive and select option 2, Save Form.
8. When the default file name (now COSTCOMP) appears, press **[RETURN]** and the form will be saved for future use and modification on your *FormFlex* work disk.

File Name Extensions

The Apple version uses the following file name extensions to differentiate the data files created by *FormFlex*:

A raw form	.RAW
An evaluated form	.FRM
Field data	.DAT
Form saved as ASCII	-none used-

Speeding Up Disk Access

It is possible to enter any character on a form, but the BASIC **INPUT** statement cannot successfully read a comma or a colon that has been written to disk. We could have circumvented this problem using the **GET** command, but disk access would have been too slow. To remedy this situation, we built a special machine-language routine into *FormFlex* that allows the reading of any text character from disk at the speed associated with the **INPUT** statement. For a detailed discussion of how this routine works, see the Apple Tech Note in this Volume.

Figure 3. Evaluation Errors

Error Message	Description
Field must have a name	There are no characters to the left of a field to identify it.
Mixed field characters	You have attempted to mix two or more field types inside one field.
Duplicate field name	You have two or more fields with the same name.
Too many fields	You have more than 80 fields.
*Field too long**	A field's length exceeds 25 characters.
*Field name too long**	A field's name exceeds 15 characters.

*Atari, Commodore 64, and TI-99/4A versions only

by 8 underlines on the sample printout) representing the field identified by its name.

Design & Evaluate

As you design a form, you can enter anything you please. When you escape from the Editor to return to the Main menu, however, (see your Control Capsule for the appropriate keypress) the computer evaluates the form. One criterion of evaluation is that each field *must* have a field name.

There is another restriction on any form you design: Due to memory limitations, the form may have no more than 80 fields. To ensure that each field in a form is identifiable, each must have a *unique* field name. If two fields have identical field names, you will receive a "Duplicate field name" error upon evaluation.

A field name can be any series of characters. Figure 2 shows the different field types and lists the restrictions for length of field names and fields for each computer. The field name may contain spaces, but if the computer encounters 2 spaces in a row, the characters to the left of the spaces are not considered part of the field name. Likewise, the field name cannot have more than one space separating it from its field—but a space is *not* required. For instance, the numeric field at the beginning of row 27 (field name "A1 $") has no intervening space between the name and the field.

Each "field name/field" combination is restricted to a single row on the form—i.e., a field name must start on the same line as the field, and the field cannot go beyond the 80th column and wrap to the next line. Also, you cannot mix field types. For example, a field is either Alpha or Uppercase. Of course, because an Alpha field can contain any printable character, it could contain uppercase characters—but you cannot *force* part of a single field to be uppercase, and allow part of the same field to be any other printable character.

The Two Phases Of Evaluation

After you have completed designing your form, you can evaluate it by pressing the key that returns you to the main menu. Evaluation is a two-phase process. The first phase encompasses the rules defined above. (See Figure 3 for a complete list of possible errors in phase 1.) The second phase allows you to define characters for Edit Character fields and Formulas. The number of characters you can specify for an Edit Character field or within a Formula is limited by the space allowed in your computer's input box at the top of the screen.

A good example of an Edit Character field is the "Tel:" field of our COSTCOMP form. If you load this form into the *Use Form* program, you will only be able to enter the characters used to depict telephone numbers—i.e., 1234567890()-

Formula Hints

A *FormFlex* formula is a mathematical expression that can contain parentheses, digits, decimal points, the 4 major math operators (+, −, *, and /) and variables (which are complete field names). The precedence of operators in a *FormFlex* Formula is the same as in BASIC: Multiplication and division are done first, and then addition and subtraction. You can alter the order of precedence by using parentheses.

There are some things that you should consider when using the *FormFlex* formula feature. First off, the field names should be kept short. Because the length of a formula is limited to the size of the input box, the shorter the field names, the more complex formulas you can enter.

Try to avoid using numbers and operators (+,−,*, and /) within a field's name. This will only confuse you and the computer as to what is supposed to be calculated for a formula. For example, what if you had a field with the name 12, and one of your formulas is 12/2? Because *FormFlex* evaluates all field names before it evaluates numeric quantities within a formula, it would interpret the 12 as a field name and *not* as the number 12.

Formulas are calculated in the order in which they appear on a form—going left to right, and then top to bottom. Because of this, you should avoid creating Formula fields that depend upon a value from a *subsequent* Formula field in its calculation.

Other Editor Features

The *Make Form* Editor is a miniature word processor. Although it does not have fancy features (like cut and paste, merge, or print formatting for bold, italic, etc.), it is an easy-to-use text editor in which each location on the piece of paper is accurately identifed by the Row and Column indicator (found in the upper-left corner of the screen). Your control Capsule carefully details all of the editing keypresses.

For added convenience, the Editor allows you to Save and Load "raw" or unevaluated forms. You can use the same name for a raw or an evaluated form because the program adds an extension on to the name for each type of file so it can tell them apart. (See Figure 4 for a chart of the different types of files that are created using the *FormFlex* programs.) This means that you can work on a form in several short sessions before it is evaluated—by Loading and Saving the raw form until you are ready to go through the evaluation process. You can also erase any form from the computer's memory with a simple keypress. (Don't worry, if you press the Erase Form key by mistake, you are given a chance to change your mind.) If you want to return to the Main Menu without evaluating a form, first Save the raw form to disk, then Erase the form from memory, exit the Editor, and no Evaluation will occur.

Main Menu Revisited

The *Make Form* Main Menu allows you to access routines to alter and print out a fully evaluated form. You can Erase the form using option 2. You can change any formula or alter the characters that will be acceptable in an Edit Character field using option 3. Disk Routines (option 4) were explained above.

If you select option 5—Printer Routines—you are presented with this menu:

```
Printer Routines

1) Print Form
2) Print Field Specifications
3) Return to Make Form Menu
```

The Print Form option lets you get a printout of the form, and puts blanks (underlines) where all the field data will go. If you just want to make up a quick master form for taking notes, this printout will suffice. If, however, you want to produce a form that allows you to enter data via the computer into a set of related forms, then this printout will just be a hardcopy master for the filled-out forms you create in the *Use Form* program.

The Print Field Specifications option prints out a list of all the field information. The field specification provides you with the field name, the field type, the field length, the row where the field appears on the form, and the column where the field begins in that row. For example, the field specification for the "Project No.:" field is

```
Field name: Project No.:
Field type: NUMERIC
Field Length: 8
ROW: 4  COL: 19
```

SPECIAL HINTS

Making an Atari *FormFlex* Work Disk

First use DOS 2.5 to format a disk to be your *FormFlex* work disk. Then use DOS to copy the following files from your HCJ Volume 2 disk to the newly formatted *FormFlex* work disk:

```
FORMFLEX
MAKEFORM
USEFORM
COSTCOMP.FRM
```

File Name Extensions

The Atari version uses the following file name extensions to differentiate the data files created by *FormFlex*:

A raw form	.RAW
An evaluated form	.FRM
Field data	.DAT
Form saved as ASCII	-none used-

Making Editing A 'Joy'

This Volume's Atari Tech Note presents a routine that allows you to control the computer's cursor with a joystick. If you run this program prior to running *FormFlex*, you can use a joystick to move the cursor while editing a form.

Now, how about making the joystick control the Page left, right, up and down functions? This can be accomplished fairly easily: Right after you run the BASIC file JOYCSR, execute the following **POKE**s:

POKE 1788,130:POKE 1789,128:
POKE 1790,138:POKE 1791,136

Now, run *FormFlex* and start editing a form. You will find that moving the joystick left, right, up and down has the same effect as pressing the page left, right, up and down keys. To understand why this works, read this Volume's Atari Tech Note.

If the field is a Formula or Edit Characters field, the field specification also includes the formula or Edit Characters for that field. For example, the field specification for the "Aav$" field is

```
Field name: Aav$
Field type: FORMULA
Formula: (A1 $ + A2 $ + A3 $ + A4 $)/4
Field Length: 8
ROW: 27  COL: 61
```

Use Form

Once you have completed a master form design, you are ready to select *Use Form* program from the main menu so you can fill out the form. Any master form that you design using *Make Form* can be loaded into *Use Form* and filled out. Then, that particular set of field data can be saved to its own data file. You may then clear the field data, fill the form out again with new data, and save it to its own file. By repeating this process, you can create any number of filled-out forms—each saved in its own data file.

In our Cost Comparison form example, we can load the COSTCOMP file into memory, then fill it out with data for a paricular job, and save it with a file name relating to that job. In short, the COSTCOMP master form becomes a template for any forms we want to fill out with that type of information.

To run *Use Form*, select option 2 from the *FormFlex* Main Menu. If you are using *Make Form*, you must first select option 6 to return to the *FormFlex* Main Menu.

When you select *Use Form*, it is loaded from disk and run. You are first asked to enter a file name. This is the name of a form that you created using *Make Form*. The reason *Use Form*

Formulating Formulas For Forms

Below are two formulas that are used in our sample form, COSTCOMP. The first formula calculates the average quoted price for Item A. The result is stored in the "Aav$" field. The second formula calculates the total for high quotes. Notice how field names such as "A1 $" and "High A $" are used in the formulas.

Formula for the "Aav$" field:

```
(A1 $ + A2 $ + A3 $ + A4 $)/4
```

Formula for the "High Total $" field:

```
High A $ + High B $
```

Hint: To keep calculating time to a minimum, try to use formulas sparingly, and limit the total number of fields in your form. Calculating time ranges from especially slow on the TI and Atari versions, to fairly slow on the C-64 and Apple versions, to instantaneous on the compiled IBM version.

SPECIAL HINTS

Making a C-64 *FormFlex* Work Disk

Format a disk using the following command:

```
OPEN15,8,15,"N0:FORMWORK,V2":CLOSE15
```

Then Copy the following files to the newly formatted disk using any file-copy program such as COPY/ALL (on the Test/Demo disk that comes with the1541 disk drive), or UNI-COPY or SD.COPY .C64 (on the Test/Demo disk that comes with the 1571 disk drive).

```
formflex
makeform
useform
boot.form
master64I10u-a
master64I10u-b
costcomp.frm
```

File Name Extensions

The Commodore version uses the following file name extensions to differentiate the data files created by *FormFlex*:

A raw form	.RAW
An evaluated form	.FRM
Field data	.DAT
Form saved as ASCII	-none used-

Quicker Editing

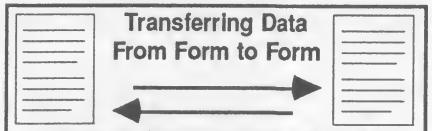
The C-64's editor allows you to shift the screen horizontally across a form one character at a time. When you attempt to type off the right edge of the screen, for example, your view of the form shifts, allowing you to see where you are typing. Most of the time, this capability is very useful, but having to wait for the screen to update with every keypress can really slow you down. When this occurs, we suggest that you press **f5** (to page right) before attempting to type any further. This will speed up the creation of forms immensely.

Master-64

The Commodore 64 version of *FormFlex* was written in a programming language called Master-64, available from Abacus Software. Master-64 adds several commands to the 64's built in BASIC. When you first run *FormFlex*, Master-64's run-time package (contained in the files master64I10u-a and master64I10u-b) is loaded from disk to install the added commands.

In order to edit the *FormFlex* program files, you need to own Master-64, as the run-time package does not allow program modification. Normally, you are not even able to list the program without owning Master-64. We have, however, discovered a loophole: To list a *FormFlex* program using the run-time package, first exit *FormFlex* by choosing option three from the Main Menu. Now, load in the Master-64 program that you wish to list (i.e., makeform, useform, or boot.form). Finally, to list the program, enter **FIND @@**. The program will now list to the screen.

Transferring Data From Form to Form



UseForm's Load Field Data option allows you to transfer field data from one form to another. As long as the data being loaded comes from a field with the same name as a field in the present form, the data is transferred. If the data being loaded originates from a field that does not have the same name as a field in the present form, the data is ignored.

For example, let's say you have two separate forms, each with a "Name" and an "Address" field. If you fill out one form and save the field data, and then load this same data into the second form, the "Name" and "Address" data are transferred to the second form.

begins by prompting you for a file name is that you must load a form created by *Make Form* to use any of *Use Form's* operations. Thus, to enter data into the Cost Comparison form, enter the file name COSTCOMP. After you enter the name, the file is loaded and you are presented with the *Use Form* Main Menu:

```
UseForm

1) Fill Out Form
2) Clear Fields
3) Disk Routines
4) Printer Routines
5) Return to FormFlex Menu
```

To fill out the form, select option 1. The *Use Form* Fill Out Form option is designed specifically to enter and edit all the field data on a form. You cannot access or change any of the field names or any other text in the form—that can only be done from *Make Form*.

The program then displays your form, with all fields blank. The cursor is placed in the first field on the form, ready for your input. You can enter any legal characters for that field. In our Cost Comparison form, for example, the cursor will be at the beginning of the "Project No.:" field. Because this is a numeric field, you are only able to enter digits, decimal points, or minus signs. When you exit the field (see the Control Capsule for the appropriate keypress for Fill Out Form mode), the number will be formated to the proper number of decimal places. In the case of the "Project No.:" field, if you do insert a decimal point when you move to the next field, the number is rounded to the nearest whole number, and no decimal point is displayed. This is because no decimal point was specified when the form was created using *Make Form*.

By using the appropriate keys, you can edit all fields in the form except Formula fields, which are calculated by the computer upon request (see Contol Capsule). If your form has very many fields or complex formulas, this figuring process can take a long time. Thus, it is a good idea to only request the computer to calculate the formulas when you have completely finished filling in the other fields. (See previous sidebar, "Formulating Formulas for Forms.")

There are several keypress functions available in Fill Out Form mode that allow you to save and load the field data to and from disk, get a printout of the form, clear the fields of all data, etc. Check your Control Capsule for a compete list of what is available on your computer. These functions are for your convenience, so you don't have to exit to the Main Menu to do general housekeeping chores.

Disk Routines

When you save data from *Use Form*, the program saves the field data associated with each field name. Because you will probably want to have several different filled-out forms for each form that you create in *Make Form*, the name for each *Use Form* data file should be different from the name of the master form created with *Make Form*.

A handy feature of *Use Form* allows you to save any filled-out form as an ASCII text file, and then load it into your favorite word processor (if it can import straight ASCII files). Just select this option from the Disk Routines menu, supply a file name, and the file will be saved.

Printer Routines

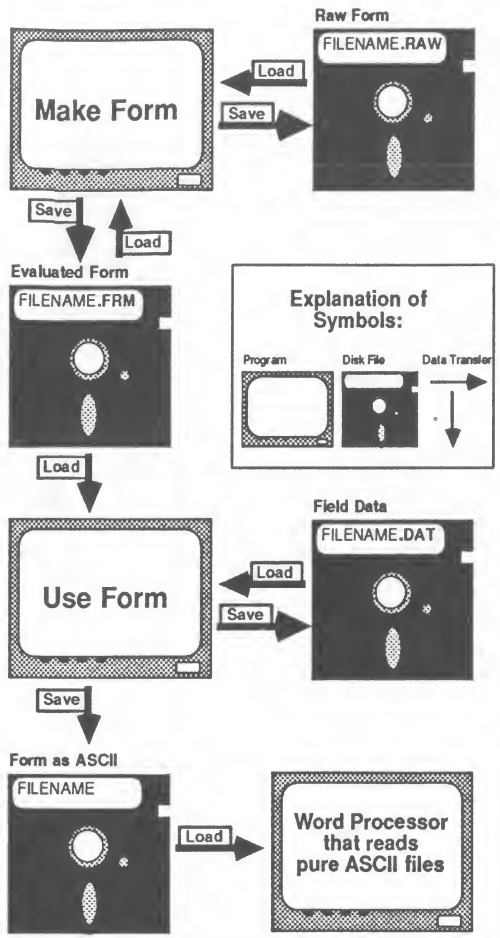
When you select the Print Routines option from the *Use Form* main menu, you are presented with this menu:

```
Printer Routines

1) Print Filled Out Form
2) Print Field Data
3) Return to Use Form Main Menu
```

Option 1 allows you to get a printed copy of the complete form, with all field data in place. Option 2 sends all field names and the associated field data of the form to the printer.

Figure 4. *FormFlex* File Structure



Note: *See next page for the appropriate Control Capsule for your machine.*

SPECIAL HINTS

Making an IBM *FormFlex* Work Disk

1. Place your DOS master disk (hereafter referred to as the DOS disk) in drive A: and turn on the power to your system.
2. Enter the command **FORMAT B: /S /V**
3. The computer will ask you to place a blank disk into drive B: to be formatted. Ensure that the blank disk is in the drive and then press [**Enter**]. After formatting, you will be asked for a volume name. Enter the name **FORMWORK**. Then, you will be asked if you want to format another. Respond "No" to this prompt to return back to DOS.
4. Place the HCJ Volume 2 disk into drive A: and your newly formatted FORMWORK disk in drive B: (If you only have one disk drive, when you enter each of the following commands, you will be prompted to insert disks for drive A: and drive B: in turn—the HCJ disk will be the disk for A: and the FORMWORK disk will be the disk for B:)
5. Enter the following commands:

```
COPY A:FORMFLEX.COM B:
COPY A:MAKEFORM.CHN B:
COPY A:USEFORM.CHN B:
COPY A:COSTCOMP.FRM B:
COPY A:COSTCOMP.FLD B:
```

6. After the last file (COSTCOMP.FLD) has been copied, remove both disks from the system.

Label the new disk as FORMWORK. The new disk that you have created can now be used to boot your system. To run *FormFlex*, enter FORMFLEX at the A> prompt.

File Name Extensions

The IBM version uses the following file name extensions to differentiate the data files created by *FormFlex*:

A raw form	.RAW
An evaluated form	.FRM and .FLD (See below)
Field data	.DAT
Form saved as ASCII	.TXT

Two Files In One

When the IBM version saves an evaluated form, it saves the form as two separate files. One file is saved with the FRM extension and one with the FLD extension. The first file (FRM) comprises the basic layout of the form. The second file (FLD) contains all the field specifcations such as the field type and field length. If you want to copy an evaluated form file from DOS, you must copy both the FRM and FLD files. To copy an evaluated form using *FormFlex*, simply load the evaluated form from *Make Form's* Disk Routines menu and then save it off to a separate disk.

SPECIAL HINTS

Making a TI *FormFlex* Work Disk

Format a disk giving it the name FORMWORK using the Disk Manager command module. Then use the Copy Files option to copy the following files to the newly formatted disk.

```
FORMFLEX
MAKEFORM
USEFORM
EDITOR_OBJ
COSTCOMP_F
```

File Name Extensions

The TI version uses the following file name extensions to differentiate the data files created by *FormFlex*:

A raw form	_R
An evaluated form	_F
Field data	_D
Form saved as ASCII	-none used-

Incorporating *HCJWord*

This Volume's TI Focus presents a word processor called *HCJWord*. This word processor is the perfect companion for *FormFlex*, because a form that is saved as ASCII from *Use Form's* Disk Routines' menu can be directly loaded, edited, and printed by *HCJWord*. Also, raw forms saved from *Make Form's* editor are compatible with *HCJWord*. Just remember to add the _R extension to the filename when loading a raw form into *HCJWord*.

FormFlex

Apple Make Form's "Edit Form" mode:

KEY	FUNCTION
Open ♦ A	Enter an Alpha field
Open ♦ U	Enter an Uppercase Alpha field
Open ♦ E	Enter an Edit Characters field
Open ♦ N	Enter a Numeric field
Open ♦ F	Enter a Formula field
Open ♦ D	Enter a decimal place for a Numeric or Formula field
TAB	Tab right 8 spaces
Open ♦ TAB	Tab left 8 spaces
RETURN	Move down one line and all the way to the left
Open ♦ I	Insert a character
Open ♦ CSR ↑	Page up
Open ♦ CSR ↓	Page down
Open ♦ \	Insert a line
Open ♦ Delete	Delete a line
Open ♦ S	Save raw form
Open ♦ L	Load raw form
Open ♦ P	Print raw form
Open ♦ C	Erase form
Escape*	Exit Edit Form
CTRL X	Exit Edit Form (UnEnhanced IIe)

Apple Use Form's "Fill Out Form" mode:

KEY	FUNCTION
RETURN	Move to next field
CSR ↓	Move to next field
CSR ↑	Move to previous field
Open ♦ CSR ↑	Move to first field
Open ♦ CSR ↓	Move to last field
Open ♦ F	Perform calculations
Open ♦ Delete	Clear current field
Open ♦ C	Clear all fields
Open ♦ P	Print form
Open ♦ O	Print fields only
Open ♦ S	Save field data
Open ♦ L	Load field data
Escape*	Exit Form
CTRL X	Exit Form (UnEnhanced IIe)

* Do **NOT** use the Escape key on an unEnhanced Apple IIe.

Atari Make Form's "Edit Form" mode:

KEY	FUNCTION
CTRL A	Enter an Alpha field
CTRL U	Enter an Uppercase Alpha field
CTRL E	Enter an Edit Characters field
CTRL N	Enter a Numeric field
CTRL F	Enter a Formula field
CTRL D	Enter a decimal place for a Numeric or Formula field
TAB	Tab right 5 spaces
SHIFT TAB	Tab left 5 spaces
RETURN	Move down one line and all the way to the left
CTRL O	Page up
CTRL P	Page down
CTRL L	Page left
CTRL ;	Page right
SHIFT INSERT	Insert a line
SHIFT DELETE	Delete a line
CTRL M	Load raw form
CTRL ,	Save raw form
CTRL .	Print raw form
SHIFT CLEAR	Erase form
ESC	Exit Edit Form

Atari Use Form's "Fill Out Form" mode:

KEY	FUNCTION
RETURN	Move to next field
CSR ↓	Move to next field
CSR ↑	Move to previous field
CTRL O	Move to first field
CTRL P	Move to last field
CTRL C	Perform calculations
SHIFT CLEAR	Clear current field
CTRL M	Load field data
CTRL ,	Save field data
CTRL .	Print form
CTRL F	Print fields only
ESC	Exit Form

C-64 Make Form's "Edit Form" mode:

KEY	FUNCTION
CTRL A	Enter an Alpha field
CTRL U	Enter an Uppercase Alpha field
CTRL E	Enter an Edit Characters field
CTRL N	Enter a Numeric field
CTRL F	Enter a Formula field
CTRL D	Enter a decimal place for a Numeric or Formula field
CTRL I	Tab right 5 spaces
RETURN	Move down one line and all the way to the left
f1	Page up
f7	Page down
f3	Page left
f5	Page right
f2	Insert a line
f4	Delete a line
f6	Save raw form
f8	Load raw form
CTRL P	Print raw form
SHIFT CLR	Erase form
CTRL X	Exit Edit Form

C-64 Use Form's "Fill Out Form" mode:

KEY	FUNCTION
RETURN	Move to next field
CSR ↓	Move to next field
CSR ↑	Move to previous field
f1	Move to first field
f7	Move to last field
f3	Perform calculations
SHIFT CLR	Clear current field
f5	Clear all fields
f2	Print form
f4	Print fields only
f6	Save field data
f8	Load field data
CTRL X	Exit Form

IBM Make Form's "Edit Form" mode:

KEY	FUNCTION
Ctrl A	Enter an Alpha field
Ctrl U	Enter an Uppercase Alpha field
Ctrl E	Enter an Edit Characters field
Ctrl N	Enter a Numeric field
Ctrl F	Enter a Formula field
Ctrl D	Enter a decimal place for a Numeric or Formula field
Tab	Tab right 5 spaces
Shift Tab	Tab left 5 spaces
Enter	Move down one line and all the way to the left
Pg Up	Page up
Pg Dn	Page down
Ctrl Pg Up	Move to top of form
Ctrl Pg Dn	Move to bottom of form
F1	Insert a line
F2	Delete a line
F6	Save raw form
F7	Load raw form
F8	Print raw form
F10	Erase form
Esc	Exit Edit Form

IBM Use Form's "Fill Out Form" mode:

KEY	FUNCTION
Enter	Move to next field
CSR ↓	Move to next field
CSR ↑	Move to previous field
Pg Up	Move to first field
Pg Dn	Move to last field
F1	Perform calculations
F2	Clear current field
F10	Clear all fields
Esc	Exit Form

TI Make Form's "Edit Form" mode:

KEY	FUNCTION
Ctrl A	Enter an Alpha field
Ctrl U	Enter an Uppercase Alpha field
Ctrl E	Enter an Edit Characters field
Ctrl N	Enter a Numeric field
Ctrl F	Enter a Formula field
Ctrl D	Enter a decimal place for a Numeric or Formula field
ENTER	Move down one line and all the way to the left
FCTN 1	Delete characters
FCTN 2	Insert characters
FCTN 3	Delete line
FCTN 4	Roll down
FCTN 5	Next window
FCTN 6	Roll up
FCTN 7	Tab
FCTN 8	Insert line
CTRL 1	Save raw form
CTRL 2	Load raw form
CTRL 3	Print raw form
CTRL 4	Erase form
FCTN 9	Exit Edit Form

Entering Formulas or Edit Characters:

FCTN X	Move to next line
FCTN E	Move to previous line
ENTER	Accept entry

TI Use Form's "Fill Out Form" mode:

KEY	FUNCTION
ENTER	Move to next field
FCTN E	Move to previous field if you are *not* on the first field
FCTN E	Calculate Formula fields if you *are* on the first field
FCTN X	Exit Form

Re-Weaver

Re-Weave your WordWeave files your way and print out a customized masterpiece.

So, with the help of *WordWeave*, you've created the ultimate who-dun-it, complete with a two-bit, unshaven, back-alley gumshoe. It's packed with intrigue, adventure, and romance. And after hours of experimentation on how the story should flow, you've found the perfect path (or trail). Now, how about making a final printout so that you can give a copy to your friends, or maybe even to your publisher?

Although, *WordWeave* does provide a Print option, it doesn't print the pages in the order in which the story is read. Besides that, the printout is single-spaced and only as wide as your computer's screen. Ideally, you want a printout with the line spacing and margin settings that *you* select. *Re-Weaver* to the rescue.

Re-Weaver is a trail-marker/text-formatter for use with *WordWeave* text files. With it, you can print your story using the trail that you define, and the printer formatting options that you select. Each trail that you create can be saved for later use or editing. The printer formatting options (Print Parameters) include margin settings, line spacing, lines per page, page breaks, page numbers, and selective printing of your *WordWeave* Header and/or Branch text. Also, when you save your trail, the Print Parameters are automatically saved along with it.

Note: Before you can use this program, you must have created a *WordWeave* data disk with at least one page of text on it. We also recommend that you prepare a separate formatted disk to save your trail and Print Parameters. Do *not* use the *WordWeave* disk for saving trail data. Doing so will limit the number of pages in your *WordWeave* file and possibly destroy valuable data.

The Main Menu

The program's Main Menu presents you with the following 6 options:

```
1.  Make Trail
2.  Read Trail
3.  Save Trail
4.  Load Trail
5.  Print Trail
6.  Exit
```

1. Make Trail

A *WordWeave* story can be read in almost any order, branching from one page to another (i.e., page 1. . . page 7. . . page 2. . .). When you choose this option, you are marking a trail through your story that determines the order in which the pages will be printed. Before choosing this option, you must place a *WordWeave* data disk into the disk drive. If you don't, an error message results, and you'll be brought back to the Main Menu.

Make Trail mode is identical to *WordWeave*'s Read mode. You can display the Branches menu, and select an option, go back to the previous page, or return to the Main Menu. There is a different keypress for each of these options. See your computer's Control Capsule for a complete list.

As you read through your story, the computer keeps track of the trail that you are blazing. Every time that you select a new branch, that decision is stored in the computer's memory. If you make a mistake, and choose the wrong branch, you can correct it: By using the option to return to the previous page, you can force the computer to forget that you were ever at the page from which you returned. This provides a very useful method for backtracking through your trail.

2. Read Trail

Here, you can test-drive the trail that you've created with the Make Trail option. Say that you load a trail that was saved a few weeks ago and you want to see how the story reads. To do this, simply choose the Read Trail option and you can "walk" down the story's trail, page by page. Again, you must insert your *WordWeave* disk into the drive before selecting this option.

In this mode, you can't decide which Branch to take—that choice was determined when the trail was created with the Make Trail option. So, to move through the story, just press the **[SPACE BAR]**. You can also go back to the previous page or return to the Main Menu. See your computer's Control Capsule for these keypresses.

If, while you are reading through your trail you find a mistake, there is a way to correct it. By pressing **[CTRL] E** from Read Trail mode, you can enter Make Trail mode and edit the trail from the current page. From here on out, you are in Make Trail mode. Any trail that was ahead of you has been

Note: *The Re-Weaver program is an enhancement to WordWeave, which appeared in HCJ Volume 1. WordWeave is a specialized word processor for creating a branching text stream—be it interactive fiction or a host of multi-path text applications for such things as adventure games, user's manuals, workbooks, and tutorial materials.*

forgotten by the computer. You may still, however, backtrack your trail.

3. Save Trail

Once you have found a trail that traverses well through your story, you may save it to disk. Be sure to save your trail on a disk other than your *WordWeave* data disk. When you select this option, the computer prompts you for a file name. If you stumble into this option by mistake, simply press [RETURN] or [ENTER] without entering a file name, and you will return to the Main Menu.

When entering a file name, we suggest that you use the title of your story. This way, you will not be confused as to which trail file corresponds with which story. After entering the file name, the computer saves your trail to disk, complete with Print Parameters (described later).

4. Load Trail

To load a trail that has been previously saved, select this option. The program will prompt you for a file name, and then load the selected trail file. If you do not enter a file name, you are returned to the Main Menu.

5. Print Trail

This option presents you with these three choices:

```
1. Change Print Parameters
2. Print it
3. Return to the Main Menu
```

Change Print Parameters

At the bottom of the Print Trail menu screen is a list of your Print Parameters and their defaults. This list is reproduced below:

```
Left margin: 5      Right margin: 75
Line spacing: 2     Lines per page: 55
Page breaks: No     Page numbers: Yes
Header text: No     Branch text: No
```

Choosing the Change Print Parameters option allows you to change the above parameters to your liking. The left and right margins can be set anywhere within 79 characters as long as the left margin is at least 5 characters less than the right margin. (For example, you may not have a left margin of 5 and a right margin of 9.) A left margin setting of 5 indents all text 5 spaces from the left of the page. A right margin setting of 75 keeps the characters 5 spaces from the right of the page on a 80-column printer. Line spacing may be 1 (single spaced) or 2 (double spaced). You can also choose the number of lines that you want printed per page. Lines per page can vary from 10 to 64.

There are probably places within your text, like the beginning of a new chapter, that you would like to have printed at the top of a new page. To do this, you need to do two things: First, you must use *WordWeave*'s Write mode to go back into your story and add the caret character ^ (the up arrow character on the Commodore 64), every place that you want the text to begin a new page. This character is used as a control character, informing *Re-Weaver*'s print routine to begin a new page. The second thing that you must do is say "yes" to the Print Parameters' page-break option. If you do not, the caret will be printed out like any other character.

If you want your pages to have page numbers printed at the bottom, say "yes" to the page numbers parameter. These numbers will be the page numbers of your trail (corresponding to the number of paper pages created)—not necessarily the page numbers of *WordWeave* screens within the trail.

When you are writing a *WordWeave* story, you can enter a Header for each page of text. If you want this Header to appear on the *Re-Weaver*'s printout, say "yes" to the Header text option. Similarly, *WordWeave* allows one line of text to

CONTROL CAPSULE

Re-Weaver

Make Trail Mode

KEY	FUNCTION
1	Choose Branch 1
2	Choose Branch 2
3	Choose Branch 3
4	Choose Branch 4
ESC	Go to previous page
CTRL X	Exit to Main Menu
CTRL B	Toggle Branches menu

Read Trail Mode

KEY	FUNCTION
SPACE	Go to next page
ESC	Go to previous page
CTRL X	Exit to Main Menu
CTRL B	Toggle Branches menu
CTRL E	Enter Make Trail mode

CONTROL CAPSULE

Re-Weaver

Make Trail Mode

KEY	FUNCTION
1	Choose Branch 1
2	Choose Branch 2
3	Choose Branch 3
4	Choose Branch 4
ESC	Go to previous page
CTRL X	Exit to Main Menu
CTRL B	Toggle Branches menu

Read Trail Mode

KEY	FUNCTION
SPACE	Go to next page
ESC	Go to previous page
CTRL X	Exit to Main Menu
CTRL B	Toggle Branches menu
CTRL E	Enter Make Trail mode

CONTROL CAPSULE

Re-Weaver

Make Trail Mode

KEY	FUNCTION
F1	Choose Branch 1
F3	Choose Branch 2
F5	Choose Branch 3
F7	Choose Branch 4
←	Go to previous page
CTRL X	Exit to main menu
CTRL B	Toggle Branches menu

Read Trail Mode

KEY	FUNCTION
SPACE	Go to next page
←	Go to previous page
CTRL X	Exit to Main Menu
CTRL B	Toggle Branches menu
CTRL E	Enter Make Trail mode

CONTROL CAPSULE

Re-Weaver

Make Trail Mode

KEY	FUNCTION
Fn 1	Choose Branch 1
Fn 2	Choose Branch 2
Fn 3	Choose Branch 3
Fn 4	Choose Branch 4
ESC	Go to previous page
Fn 7	Exit to Main Menu
Fn 8	Toggle Branches menu

Read Trail Mode

KEY	FUNCTION
SPACE	Go to next page
ESC	Go to previous page
Fn 7	Exit to Main Menu
Fn 8	Toggle Branches menu
CTRL E	Enter Make Trail mode

describe a Branch in the Branch menu. If you want this line to appear on *Re-Weaver*'s printout, say "yes" to the Branch Text option. The Branches that are printed are the ones that are selected in Make Trail mode.

Print It

Choosing this option requires that you already have a trail in memory. You can achieve this by loading a trail from disk, or choosing the Make Trail option from the Main Menu. Make sure that the *WordWeave* disk is in the drive before choosing this option.

This option prints your *WordWeave* story, using the trail and Print Parameters that you have selected. For best results, align the printer paper's perforation just above the print head.

The Rules

Because your computer's screen width will not always match the width specified by the Print Parameters' margin settings, *Re-Weaver* performs automatic wordwrap whenever it prints a trail. To wrap words correctly, *Re-Weaver* follows a simple set of rules. You should be aware of these rules when using *WordWeave*, so that *Re-Weaver* will correctly format your text. When reading these rules, keep in mind that there are two types of pages: There's the *WordWeave* page that is created by the *WordWeave* program, and there's the page of formatted text that is produced by *Re-Weaver*'s print routine. You see, one *Re-Weaver* page may consist of several *WordWeave* pages, depending on the trail chosen and the Print Parameters used. The following figures (labeled Rules 1 through 6) illustrate each rule and their function.

Rules 1 and 2 take care of most of the word wrapping. Pay particular attention to Rule 2. Because of Rule 2, you must never let a word *break* (a word that wraps from one screen line to the next) when entering text with *WordWeave*. If you do, *Re-Weaver* will consider the broken word to be two separate words and print them as such.

CONTROL CAPSULE

Re-Weaver

Make Trail Mode

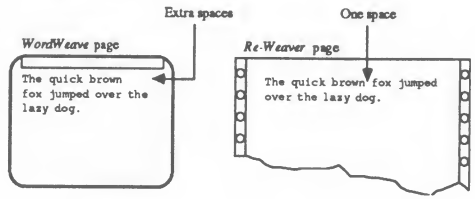
KEY	FUNCTION
1	Choose Branch 1
2	Choose Branch 2
3	Choose Branch 3
4	Choose Branch 4
FCTN 9	Go to previous page
CTRL X	Exit to Main Menu
CTRL B	Toggle Branches menu

Read Trail Mode

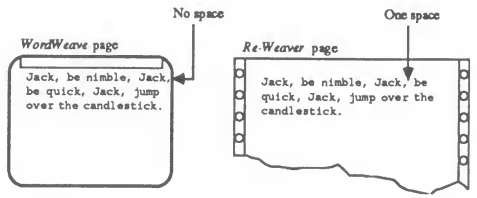
KEY	FUNCTION
SPACE	Go to next page
FCTN 9	Go to previous page
CTRL X	Exit to Main Menu
CTRL B	Toggle Branches menu
CTRL E	Enter Make Trail mode

The Rules for Text Formatting in *Re-Weaver*

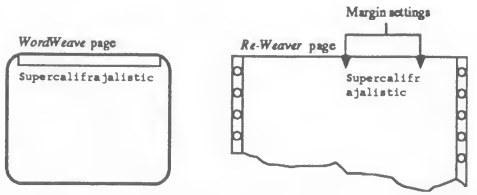
Rule 1: Spaces found at the end of *WordWeave* page screen lines are considered to be one space. This is done so that the first word on the next line can be pulled up and attached to the previous line correctly.



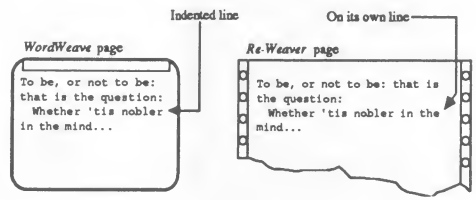
Rule 2: If there isn't a space at the end of a *WordWeave* page screen line, one is added. This is done so that if a word is pulled up from the next line, it will have a space between it and the previous word.



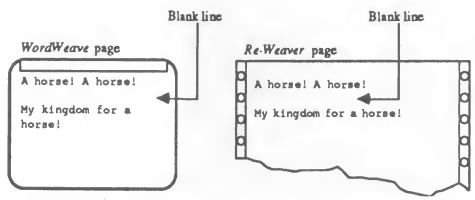
Rule 3: All words that are too long to fit within the margin settings of a *Re-Weaver* page are broken, not hyphenated.



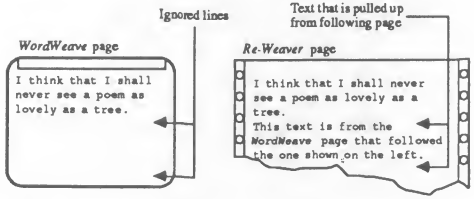
Rule 4: All indented lines of text found in a *WordWeave* page are considered to be the beginning of a paragraph. These lines will always begin on a new line within the *Re-Weaver* page.



Rule 5: All blank screen lines found within the text of a *WordWeave* page are printed as blank lines on the *Re-Weaver* page.



Rule 6: All blank screen lines found at the end of a *WordWeave* page are ignored so that the following *WordWeave* pages of text can be pulled up.



Upsets

A challenging, yet very easy-to-learn educational brain-game based on the concepts of set theory.

Set theory is important in a structured approach to mathematics, and has consequently become an integral part of many computer programming languages (e.g., Pascal, C, etc.). The *Upsets* program is an educational game that offers you an opportunity to learn about and practice the basics of set theory, and then apply your understanding in either a solitaire or two-person game. If you are new to the concepts, terms, and symbols used in set theory, first see our sidebar "Getting Set-tled For Upsets"

The game is designed around a deck of cards with four different colored dots. The four colors on the cards are the elements used to describe which cards are in the set. For example, a set may comprise all cards with a blue dot and be identified by the single element **BLUE**. In Figure 1, this would include cards 1, 3, and 4. Another set could be all cards which have either blue dot or a red dot using the union function. The set expression would be:

BLUE ∪ RED

This set would include cards 1, 3, 4, 5, and 6 in Figure 1.

Another set could be described using the intersection function to include only cards having both a blue dot and a red dot. The expression for this set would be:

BLUE ∩ RED

This set would include cards 1 and 3 in Figure 1.

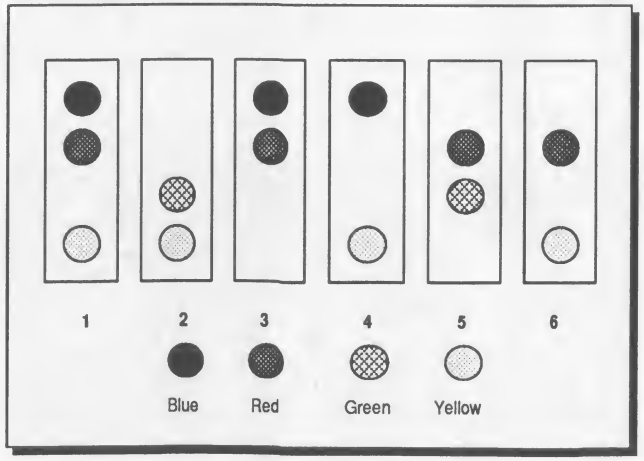
When the minus sign appears in a set expression, it will indicate the exclusion of one or more colors from the set being defined. An example using the difference connective is:

BLUE – RED

In this example, only card 4 would be selected from Figure 1. This means that you take the set of all cards with a blue dot, then remove any cards having a red dot.

Upsets displays up to four colors on each of the cards. You may use any or all of these four colors with any of the three connectives to form a set expression or set name. The set you describe could include all, some, or none of the cards—depending on the colors, connectives, and position of the parentheses.

Figure 1.
A sample set of "cards" in a game of Upsets. Note that the colors of the cards in your computer's version may be different from those indicated below.



Get Ready, Get Set . . .

The first menu in the program allows you to select one of three different games:

```
(1) Practice
(2) Solitaire
(3) Challenge
```

When you select option 1 for a Practice session, the computer does not keep score, or limit the game length. In this mode, you are able to create set names and watch the results. This provides an excellent way to discover the logic rules that govern set names. The computer places the cards on the screen randomly from a "deck" of fifteen cards. Note that a blank card is never one of the cards in the game, because it would represent the "null" set. According to the rules of set theory, the null set is included in *all* sets and, because a blank card would *always* be selected, it is not used in the game of *Upsets*.

Option 2, Solitaire, lets you play *Upsets* by yourself. Here, you express sets that include as many cards as possible on your side of the screen (the left), while selecting as few cards as possible from the right side of the screen. You make all the plays in Solitaire, but the computer keeps score, giving you 10 points for each card included in the set on your side of the screen, and giving your non-playing "opponent" 10 points for each card that your expression selects on the right. You can win the game in two ways: (1) by selecting *all* of your cards and *fewer* than all of your opponent's cards in any one turn, or (2) by having the highest score after 10 turns.

In Option 3, Challenge, two players take turns defining the set name. One player uses the cards on the left side of the screen, while the other uses the cards on the right side. After each player takes a turn, the computer evaluates the new set, and each player receives 10 points for each card in the set.

After you select a game option, a prompt asks you to enter the players' names (one or two names, depending on the game type selected). Next, you select the number of cards you would like to work with. You can have from 3 to 6 cards displayed on each side of the screen. The more cards you select, the more complicated the game, and the greater the possible score.

Finally, a prompt asks how many connectives (from 1 to 3) you would like to use in the set name.

```
Select Level Of Expression:
1) One Connective
2) Two Connectives
3) Three Connectives
```

Getting Set-tled For Upsets

Set theory is the branch of mathematics concerned with the definition and relationships between the grouping of objects, numbers . . . and well, *anything*. We say that a particular set "contains elements," very much like a box contains items. For example, a standard deck of 52 playing cards is a set of all the cards after you discard the "set" consisting of jokers and any other non-face or number cards. We can define a grouping or set that contains all the hearts. We say this set contains all cards with red hearts on them. We may do the same for cards having the red diamond, the black spade, and the black club.

We can define other sets by using "connectives" to form relationships between different sets. The basic connectives (those used in our *Upsets* program) are: **union**, **intersection**, and **difference**. Figures A, B, and C show three diagrams (called Venn diagrams) that graphically demonstrate how these functions work. The two circles, X and Y, represent two different sets. The black areas show the set created by the function.

The **union** function (see Figure A) is used to form a larger set containing elements of two smaller sets. This function is expressed by the ∩ symbol and is similar to the OR function used in BASIC. Taking our deck of cards again, we can define a set containing all red cards as being the union of hearts and diamonds, or expressed symbolically:

red cards = hearts ∪ diamonds

Obviously, this function combines two of our first four sets into another set of *all* red cards.

The **intersection** function is expressed by the ∩ symbol and is similar to the BASIC **AND** function (see Figure B). It creates a set by selecting elements that are contained in both sets. The fewer elements that are *shared*, the more *exclusive* the set becomes; e.g., the set of cards formed by the intersection of all aces and all red cards is a set of only two cards—the ace of hearts and the ace of diamonds.

red aces=(red cards) ∩ (aces)

The **difference** function represented by a minus sign allows for the exclusion of certain elements from a set (see Figure C). Using the deck of 52 cards, we can form a selection of all cards not having numbers—the set of all face cards and aces. Symbolically we write:

(face cards ∪ aces) = (all cards) – (cards with numbers)

These functions can be combined in set expressions, just like mathematical expressions, and parentheses are used to designate the order of evaluation.

Venn Diagrams

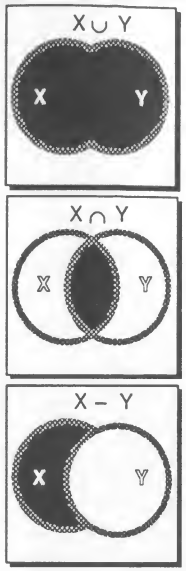


Figure A.
This diagram represents the union function—similar to the BASIC OR function.

Figure B
The intersection function is like the BASIC AND statement.

Figure C.
Here, the difference function excludes elements from set X that also occur in set Y.

This will determine the length and complexity of the set name. With only one connective, you are limited to simple set expressions like those in the examples above (e.g., **BLUE ∩ RED**). And with only one connective you may use any two (but *only* two) of the four colors in any expression.

Two connectives allow a more complex name such as: **(BLUE ∪ RED) ∩ GREEN**. Selecting three connectives expands the set name to the most complex level available in *Upsets*. It also allows the switching of parentheses to change the order in which the expression is evaluated—expanding the number of strategic tools available.

The Playing Screen

After selecting all of the options described, the playing screen appears (see Figure 2). Depending upon which game you've chosen, 3 to 12 cards are displayed across the top of the screen. No two cards are ever alike. When the computer evaluates a set expression, it indicates which cards are in the designated set by placing a mark above each of the cards that satisfies the set expression.

Below the cards are 2 small boxes labeled **Round** and **Turns**. The box labeled **Round** is activated only when you play the Challenge option. In this option, you play 4 rounds. A round is over when one player has created a set that includes all of the cards on either one or both sides of the screen, or after 20 turns. The player with the highest total score after 4 rounds wins the game. In rounds 1 and 3, the player on the left side of the screen goes first; in rounds 2 and 4, the player on the right side goes first.

The **Turns** box indicates the current turn, and is active only during the Solitaire or Challenge options. In Solitaire, the length of a game is 10 turns. In Challenge, the length of each round is 20 turns. A turn is over after *both* players have made a change to the current set expression.

Directly below the **Round** and **Turns** boxes are the 2 **Score** boxes each labeled with the player's name. In Solitaire, the score box on the left side always holds the player's score; the score box on the right contains the score of your non-playing opponent. In a Challenge game, the name of each player appears in the box with that player's score. The score displayed here is only for the current round. At the start of each round, each **Score** is added to the associated **Total** box, and the **Score** box is reset to zero.

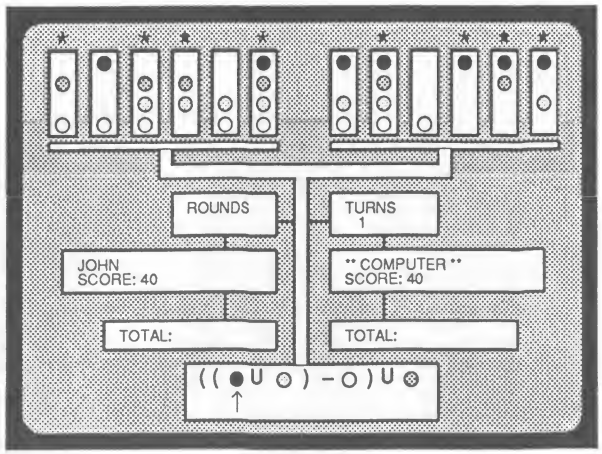


Figure 2.
This simulated screen depicts a solitaire game where the set expression describes 4 cards from each side.

Express Yourself

In the bottom center portion of the screen is the current set name. At the beginning of a game or round, the name is empty except for parentheses:

((. . .) . .) . .

Here, we are using periods to indicate what appears on the screen as empty character positions. Below this empty set name is a cursor. The current player moves the cursor to any position in the set name and, by pressing the **[SPACE BAR]**, selects *one* legal connective symbol or color dot for that position in the name. To toggle through all of the legal symbols for that position, simply continue pressing the **[SPACE BAR]**. Once you have displayed the desired symbol, both your selection and your turn are completed by pressing **[ENTER]** or **[RETURN]**. If you move away from a position without pressing **[ENTER]** or **[RETURN]**, the original symbol appearing at that location will return.

Before a set name can be evaluated by the computer, all parts of it must be filled in. It needs to contain the number of connectives you chose at the beginning of the game *and* enough colored dots to evaluate the connectives. It may take several turns before the set name is completed for evaluation. A valid name with 1 connective may look like this:

((• ∩ •) . .) . .

(Periods represent spaces, the • character represents a colored dot, and ∩ ∪ – are the connectives.)

A set name with 2 connectives may look like this:

((• ∩ •) – •) . .

A set name with all 3 connectives would look like this:

((• ∩ •) – •) ∪ •

A set name can be filled out in any order, with each player limited to setting either a single dot color or a connective during a given turn. But until the set name is completely filled out and evaluated at least once, players can't *change* any of the connectives or dots. If all 3 connectives are being used, players also have the option of changing the order of parentheses.

Order Of Evaluation

The parentheses used in a set name, as described earlier, indicate the order the connectives are to be evaluated. The computer always evaluates that part of the expression within the innermost parentheses *first*, working its way outward to the outermost connective not contained within parentheses; or, it *begins* at the left and evaluates towards the right. The default position for parentheses will always appear as in the first example shown above, which causes connectives to be evaluated from left to right. The following examples will illustrate the effect parentheses have on a 3-connective set name:

((**B** ∩ **R**) – **G**) ∪ **Y**

(Here, the letters **B**, **R**, **G**, and **Y** represent the colors Blue, Red, Green, and Yellow.) This set would include any cards containing either (1) a yellow dot, or (2) both a blue dot and a red dot, but not a green dot.

(**B** ∩ **R**) – (**G** ∪ **Y**)

This set name is the same as the one above—*except* for the position of the parentheses. It now includes cards that contain both blue and red, but not green or yellow. Because there are only four colors, and no two cards can be identical, the set named above can include only one card.

To alter parentheses, just press **P**, and the parentheses change. This allows you to switch the parentheses to any of three possible settings; the set name at the bottom of the screen will be rewritten showing the new arrangement. Once you have the parentheses set to the desired arrangement, press **[ENTER]** or **[RETURN]** to fix the setting. This completes a player's turn and begins reevaluation of the set name and an updating of the score accordingly. If you decide that you don't want to change the position of the parentheses before you press **[RETURN]** or **[ENTER]**, you can return the name to its original form by simply moving the cursor. This allows you to continue your turn to make a different change to the set name. You may make only *one* permanent change to the set name during your turn.

To quit, press **[ESC]** (or ← on the C-64 and **[FCTN] 9** on the TI-99/4A). Whether you wish to simply introduce yourself to set theory, or use your talents of logic to develop effective game strategies, *Upsets* is definitely a real setup for challenging fun.

CONTROL CAPSULE

Upsets

KEY	FUNCTION
SPACE BAR	Change symbol
ENTER or RETURN	Accept symbol in expression
P	Alter parentheses
ESC*	Return to Main Menu

* FCTN 9 on TI-99/4A, ← on C-64

Applesoft Line Input In ProDOS

Saving and loading text files from disk in Applesoft BASIC is a relatively easy task. The BASIC **INPUT** statement, however, creates a problem when trying to read strings containing commas and colons, because these characters act as delimiters or separators. The **GET** statement provides a solution, but can be excruciatingly slow when used with even moderate-sized files.

Here's a solution for ProDOS users: A short machine-language program we call LIN for Line INput. Some BASIC languages (i.e., Microsoft BASIC and TI Extended BASIC) have a **LINE INPUT** statement, where only a carriage return (ASCII 13) acts as a delimiter, thus allowing commas and colons to be read as part of a string. We have designed our new input routine around this statement, so any character written to disk using the **PRINT** statement (except a carriage return), can be read back in.

To install this routine, use the BASIC loader on your HCJ Volume 2 disk with the file name LIN.LOADER. This program stores the routine at memory location 768 ($300), and assigns the address to the variable **LIN**. After the program is run and the routine is loaded, you can use it from within a program in place of an **INPUT** statement by executing a **CALL** in this format:

CALL LIN,A$

Here, **A$** can be *any* string variable (including a string-array variable). When you execute this call, the next string in the text file will be loaded into the variable **A$**. Also, you can relocate the routine by changing the second number in the **DATA** statement in line 20020 (currently 768) to any location you choose. Be sure that the area is large enough (has at least 60 bytes of free memory), and that it will not be overwritten by your program.

To demonstrate how this routine enhances accessing disk-based text files, we've included 4 demo programs on you HCJ Volume 2 disk: OUTPUT.IT, INPUT.IT, GET.IT and LINPUT.IT. By running these programs you wil get hands-on experience with the pitfalls of normal BASIC file access and the improved performance available with the LIN subroutine.

The first demo program, OUTPUT.IT, creates a text file for us to test out the other programs. It sequentially fills the string **A$** with all the printable characters (ASCII 32 through 126), and **WRITE**s the string 50 times to the disk file named ALL.CHARS using the **PRINT** statement.

The program INPUT.IT **DIM**s (see Listing 1) the array **A$( )** to 50 and attempts to **READ** this file using the **INPUT** statement. Instead of filling the 50 elements with all characters, each element contains only the characters up to, but not including, the first comma. Specifically, it only loads ASCII characters 32 through 43 (a space followed by the !"#$%&'()*+ characters), because ASCII 44 is a comma and acts as a delimiter in the **INPUT** statement. Consequently the ASCII characters from 44 through 126 are ignored, and are thus not loaded into our array.

The program GET.IT replaces the **INPUT A$** command with an **IF-THEN** loop and a **GET** command to **GET** each character while searching for each delimiting carriage return. Although this program *does* fill the array with the correct strings, it takes a long time.

The program LINPUT.IT (see Listing 2) acts nearly as quickly as INPUT.IT, thus making it far superior to GET.IT. In addition, it gets *all* of the characters, making it superior to either of them. The only differences between this program and INPUT.IT are that LINPUT.IT does a **GOSUB 20000** to load the machine language into memory, and also uses **CALL LIN,A$(I)** in place of the **INPUT A$(I)** statement.

How It Works

Because the LIN routine makes use of the ProDOS Machine Language Interface (MLI) to read each line, it does not work in DOS 3.3. The MLI is loaded each time ProDOS BASIC.SYSTEM is run, and constitutes an easy-to-use disk-access system for assembly language programmers.

It makes use of the Applesoft **INPUT** statement, but fools the routine into using only either a carriage return or an ASCII 0 character as a delimiter. Thus, if either of these characters is encountered as a line of text is input from disk, this routine will treat them as delimiters and end the input of the string.

We've included the object file called LIN which **BLOAD**s at 768 ($300). In addition, the compete commented source code from the Merlin PRO editor for the LIN routine is included on disk as LIN.S.

There are 3 full-sized programs on you HCJ Volume 2 disk that use this routine: (1) the new, enhanced version (1.1) of *WordWeave*, (2) *Re-Weaver*, and (3) *FormFlex*. You will find that this routine greatly enhances disk access from Applesoft, and speeds up one of the computer's biggest bottlenecks.

Listing 1

```
100  REM  ************
110  REM  * INPUT.IT *
120  REM  ************
130  REM  COPYRIGHT 1986
140  REM  HOME COMPUTING JOURNAL
150  REM  VERSION 2.O
160  REM  APPLE II FAMILY APPLESOFT
170  DIM A$(50):D$ =  CHR$ (4)
180 F$ = "ALL.CHARS"
190  PRINT D$;"OPEN ";F$
200  PRINT D$;"READ";F$
210  FOR I = 1 TO 50
220  INPUT A$(I)
230  NEXT I
240  PRINT D$;"CLOSE "
250  FOR I = 1 TO 50: PRINT A$(I): NEXT
```

Listing 2

```
100  REM  ************
110  REM  * LINPUT.IT *
120  REM  ************
130  REM  COPYRIGHT 1986
140  REM  HOME COMPUTING JOURNAL
150  REM  VERSION 2.O
160  REM  APPLE II FAMILY APPLESOFT
170  DIM A$(50):D$ =  CHR$ (4)
180  GOSUB 20000
190 F$ = "ALL.CHARS"
200  PRINT D$;"OPEN ";F$
210  PRINT D$;"READ";F$
220  FOR I = 1 TO 50
230  CALL LIN,A$(I)
240  NEXT I
250  PRINT D$;"CLOSE "
260  FOR I = 1 TO 50: PRINT A$(I): NEXT
270  END
20000  READ NU,LIN 20010  FOR IT = 0 TO NU - 1: READ X: POKE
LIN + IT,X: NEXT IT
20020  DATA 60,768,32,6,227,32,190,222,32,227,223,133,133,
132,134,169,255,197,17,240,5,162,163,76,18,212,32,0,191,202,
213,190,208,21,172,219,190,169,0,153,255,1,169,0,160,2,162,0,
32,233,227,32,123,218,96,32,139,190,170,76,18,212
20030  RETURN
```

Joystick-Controlled Cursor

Here, we present a short machine-language driver that allows a joystick to control the cursor on the Atari 800 family of computers. This routine runs in the background, so you can use it to edit a BASIC program, move through *WordWeave* text, or manipulate most any program that utilizes the cursor keys.

We provide this routine as a BASIC loader on your HCJ disk under the file name of JOYCSR. To install the joystick driver, insert your HCJ disk, type **RUN "D:JOYCSR"** and press **[RETURN]**. You may also install the driver by using the *HCJ Director* program. Once installed, you can control the cursor by using a joystick plugged into port 1. Moving the joystick in any one direction will also move the cursor. The joystick's fire button acts as a [TAB] key.

The joystick machine-language driver is stored in memory at 1726-1791 ($06BE-$06FF), the top of page 6. The source code listing of this driver is on your HCJ disk. It is a pure text file that can be edited with most any text editor and assembled with most Atari assemblers. We used the Atari Program-Text Editor and Atari Macro Assembler. The file name of the source file is JOYCSR.S.

In order to "wedge" this routine into Atari's operating system, we took advantage of system timer 2 to install an interrupt. System timer 2 uses two memory locations located at 538,539 ($021A,$021B). These locations hold the time in lowbyte-highbyte format. (Note: The time kept by this clock is not stored as hours:minutes:seconds, but simply a numeric value that is decremented every 1/60th of a second.) Whenever the timer reaches a value of zero, a software interrupt occurs, causing the computer to jump to the machine-language subroutine vectored by memory locations 552,553 ($0228,$0229). Subroutines using this vector should end in a **RTS** (ReTurn Subroutine) instruction, not a **RTI** (ReTurn Interrupt). Also, because the operating system services this interrupt for you, it is not nessecary for the interrupt routine to restore the 6502's registers to their original values.

To check the position of the joystick, the joystick driver reads memory location 632 ($0278), where the computer stores the direction that the joystick is pointing. To check the fire button, the joystick driver reads memory location 644 ($0284). Once the joystick's direction and fire button status are read, we store it off for evaluation. Now, every 1/60th of a second (every time the interrupt occurs), the computer tests the joystick for one position—up, down, left, right, or fire. It takes five interrupts for all possible joystick positions to be tested. When our driver is done checking all five positions, we read the joystick status again, and the process starts over.

When we find a joystick position to be true, we must somehow fool the operating system into believing that a key is being pressed. To do so, we take the key's matrix code and store it at location 764 ($02FC). This memory location is where the keyboard handler gets its keyboard data. The number stored here represents the last key pressed. This value, however, is not ATASCII. It is the unprocessed matrix code of a key. The matrix code of a key corresponds to the key's physical location on the keyboard as opposed to its alphabetical or numeric position. By storing the matrix code of a key here, we can fool the keyboard handler into believing that a key was actually pressed.

To convert joystick positions to keyboard matrix codes, the joystick driver uses a look-up table stored in memory at 1787-1791 ($06FB-$06FF). By using the position of the joystick as a pointer, we can index into our table to get the appropriate matrix value of a key. The chart above shows the format of this look-up table. Numbers are shown in both decimal and hexadecimal.

The matrix codes that make up this table are stored in line 280 of the BASIC loader. By changing these numbers, you can change the keys that each joystick position corresponds to. For example, to make the fire button act as a [RETURN] key instead of a [TAB] key, change the 44 in line 280 to a 12 and re-run the program. Alternatively, you could execute a **POKE 1787,12** after running the BASIC loader. To make the modification of this program easier, we have provided you with a list of the different keypresses and their matrix code equivalent.

Memory Location		Matrix Code		Keyboard Key	Joystick Position
1787	$06FB	44	$2C	TAB	Fire button
1788	$06FC	135	$87	CTRL *	Right
1789	$06FD	134	$86	CTRL +	Left
1790	$06FE	143	$8F	CTRL =	Down
1791	$06FF	142	$8E	CTRL -	Up

Keyboard Matrix Codes—PEEK (764)

Key	Normal	Shift	Control	Key	Normal	Shift	Control
A	63	127	191	1	31	95	159
B	21	85	149	2	30	94	158
C	18	82	146	3	26	90	154
D	58	122	186	4	24	88	152
E	42	106	170	5	29	93	157
F	56	120	184	6	27	91	155
G	61	125	189	7	51	115	179
H	57	121	185	8	53	117	181
I	13	77	141	9	48	112	176
J	1	65	129	0	50	114	178
K	5	69	133	*	7	71	135
L	0	64	128	+	6	70	134
M	32	101	165	,	32	96	160
N	35	99	163	-	14	78	142
O	8	72	136	.	34	98	162
P	10	74	138	/	38	102	166
Q	47	111	175	;	2	66	130
R	40	104	168	<	54	118	182
S	62	127	190	=	15	79	143
T	45	109	173	>	55	118	182
U	11	75	139	SPACE	33	97	161
V	16	80	144	RETURN	12	76	140
W	46	110	174	TAB	44	108	172
X	22	86	150	ESC	28	92	156
Y	43	107	171	CAPS	60	124	188
Z	23	87	151	ATARI	39	103	167

```
100 REM ******************************
110 REM * JOYSTICK CONTROLLED CURSOR *
120 REM ******************************
130 REM COPYRIGHT 1986
140 REM HOME COMPUTING JOURNAL
150 REM VERSION 2.0
160 REM ATARI 6502 FOR THE 800, 800XL, 130XE
170 ? CHR$(125);"STORING MACHINE LANGUAGE...";
180 FOR I=1726 TO 1791:READ D:POKE I,D:NEXT I:D=USR(1726)
190 ? CHR$(125);"PORT 1 JOYSTICK DRIVER INSTALLED."
200 DATA 104,169,207,141,040,002,169,006
210 DATA 141,041,002,169,001,141,026,002
220 DATA 096,173,250,006,172,249,006,208
230 DATA 012,173,132,002,010,010,010,010
240 DATA 013,120,002,160,005,136,074,176
250 DATA 006,190,251,006,142,252,002,141
260 DATA 250,006,140,249,006,169,001,141
270 DATA 026,002,096,000,000
280 DATA 044,135,134,143,142
```

Alpha Lock

The Commodore 64's [SHIFT LOCK] key is an actual SPST (Single-Pole, Single-Throw) pushbutton switch that produces the same effect as holding down the [SHIFT] key. This is fine for typing capital letters, provided you don't make any mistakes. The problem occurs when you need to edit what you're typing. For example, pressing [DEL] while the [SHIFT LOCK] key is down doesn't delete characters—it inserts them. This is because *all* of the computer's keys have been placed in their shifted position: The cursor keys are stuck at UP and LEFT only, periods and commas become greater-than and less-than signs, and all of the number keys are now punctuation marks and symbols. Because of its drawbacks, [SHIFT LOCK] is probably the Commodore 64's least-used key.

On many computers, you have what is called an "alpha lock." An alpha-lock key is basically the same as a shift lock, except that it only affects the alpha characters (letters a through z). Through software, we have been able to create an alpha-lock key for the Commodore 64. We call this program—can you guess?—*Alpha Lock*. To use *Alpha Lock*, run the program ALPHALOCK found on your HCJ Volume 2 disk. This program is a BASIC loader that installs the machine code for our alpha-lock routine in the computer's cassette buffer located at 828 ($033C). Once this program has been run, you can toggle the alpha lock by pressing the **[CONTROL]** and **[SHIFT]** keys simultaneously.

Using Alpha Lock With Other Programs

This alpha-lock program can be used when entering computer commands or BASIC programs, but it's most useful when used with other programs—specifically word processors. We have tested *Alpha Lock* on some of our own programs and a couple of commercial word processors as well. *Alpha Lock* works fine with PaperClip from Batteries Included, and Speedscript from Compute!. When used with PaperClip, however, pressing **[CONTROL]** **[SHIFT]** to toggle the alpha lock also puts the word processor in "Control Character" mode. To exit this mode, simply press **[CONTROL]** again, and things will return to normal.

WordWeave, from HCJ Volume 1, works fine with *Alpha Lock*, provided that you make a simple adjustment. Because *WordWeave* uses the cassette buffer, *Alpha Lock* must be relocated to work properly. This is easily accomplished by changing the variable **SA** in line 180 of *Alpha Lock* to equal a new starting address. Because memory locations 50176-53247 ($C400-$CFFF) are not used by *WordWeave*, setting **SA** equal to 50176 makes *Alpha Lock* and *WordWeave* compatible. On your HCJ Volume 2 disk, we have provided the program file WWALPHALOCK with this change already made. Unfortunately, due to the lack of free memory available when the Master-64 runtime package is loaded, *Alpha Lock* does not work with *FormFlex* in this volume.

Programs that activate Commodore's key-repeating feature make toggling alpha-lock mode a little difficult. Because keys automatically repeat in this mode, you may accidentally toggle in and out of alpha-lock mode in just one keypress. In this case, we suggest that you hold down the **[CONTROL]** key and tap the **[SHIFT]** key sharply—much like hitting **[RUN/STOP]** **[RESTORE]**. This should prevent any unwanted toggling of alpha-lock mode. Overall, *Alpha Lock* works with most programs. If you are having difficulty, try relocating *Alpha Lock* using the technique described above for *WordWeave*.

The Keyboard Decode Tables

The *key* behind this program is the 64's use of "keyboard decode tables." The 64 uses a 6526 complex interface adapter to read the computer's keyboard. By reading the 6526's registers, the computer derives the matrix code of a key. This matrix code is then used to index into a keyboard decode table to find the key's ASCII equivalent. So, it is the keyboard decode table that determines what the computer considers a key's ASCII value to be.

There are four decode tables; one for normal characters, one for [SHIFT] characters, one for [CMD] characters, and one for [CONTROL] characters. What we did was create a *fifth* table—a table for alpha-lock characters. Our table is the same as the decode table for shifted characters, except that only the keys A through Z are represented as shifted.

Located in Kernal ROM is a routine that is responsible for determining which keyboard decode table to use. According to whether or not the [SHIFT], [CMD], or [CONTROL] keys are being pressed, this routine sets memory locations 245 and 246 ($F5 and $F6) to point to the appropriate keyboard decode table. Luckily, this routine is vectored through memory locations 655 and 656 ($028F and $0290). Normally, this vector points to 60232 ($EB48), but we change it to point to our alpha-lock routine instead. The alpha-lock routine's first task is to check for the [CONTROL] [SHIFT] key combination. If pressed, the alpha-lock flag is toggled on or off. If the alpha-lock flag is set and neither [SHIFT], [CMD], nor [CONTROL] keys are being pressed, we set memory locations 245 and 246 to point to our alpha-lock decode table. This, in effect, creates an alpha-lock key.

For those of you who would like to look at *Alpha Lock's* source code, it is saved as a sequential text file on your HCJ Volume 2 disk under the file name of ALPHALOCK.S. We used Commodore's Assembler Development System, but most any assembler should do.

Once you understand how *Alpha Lock* works, it's a simple matter to modify this program to create numeric keypads, Dvorak keyboards, or any setup that you want. Instead of switching in the alpha-lock decode table, switch in a table of your own making.

Get Seg

The **DEF SEG** command gives the BASIC programmer the ability to alter BASIC's Data Segment (DS) resister. For example, **DEF SEG=&H0** sets the DS register to zero, allowing access to the first 64K of memory. This command is vital to investigating and altering the computer's memory locations, calling machine-language subroutines, and **BLOAD**ing and **BSAVE**ing data. The statement **DEF SEG**, without any assignment value, restores the DS register back to BASIC's default setting.

Question: What *is* BASIC's default setting for the DS register? Unfortunately, Microsoft neglected to provide a method of *reading* the DS register. You can *change* the DS register to your heart's content, but through BASIC, there is no direct way to see what BASIC's default value for the DS register is.

Solution #1

If you are using either an IBM PC or PCjr, you can use the following PEEKs to find BASIC's DS register:

```
DEF SEG=&H50
DS%=PEEK(16)+PEEK(17)*256
DEF SEG
```

After executing this code, the variable **DS%** will be equal to BASIC's DS register. If you own an IBM compatible, however, this method may not work. The Tandy 1000's GW BASIC, for example, does not work with the above code. So, even if this method works on most machines, it does not produce clean, transportable code.

Solution #2

Because the DS register is an actual hardware register found in the 8086 family of microprocessors, it should be a simple matter to MOVe its contents into a BASIC variable. The following machine-language subroutine, entitled *Get Seg,* does just this.

```
CSEG    SEGMENT
        ASSUME   CS:CSEG
GETSEG  PROC     FAR
        PUSH BP              ;SAVE BP
        MOV BP,SP            ;SET BASE PARM LIST
        MOV DI,[BP]+6        ;GET ADR OF PARM
        MOV [DI],DS          ;PASS BACK DS REG
        POP BP               ;RESTORE BP
        RET 2                ; FAR RETURN TO BASIC
GETSEG  ENDP
CSEG    ENDS
        END
```

CALL OFS%(DS%) calls this routine. The variable **OFS%** must contain the offset of the machine-code in memory. The contents of the DS register is returned in the integer variable **DS%**.

```
20000 '
20010 ' GET BASIC'S DATA SEGMENT
20020 '
20030 RESTORE 20070:FOR I=0 TO 6:READ X:ML%(I)=X:NEXT
20040 DS%=0:OFS%=VARPTR(ML%(0))
20050 CALL OFS%(DS%):REM CHANGE TO CALL ABSOLUTE(DS%,OFS%)
WHEN USING MICROSOFT'S QUICKBASIC COMPILER 2.0
20060 RETURN
20070 DATA -30379,-29467,-29736,1662,1417,-13731,2
```

There are several ways to put *Get Seg* in memory. The program listed above (file name GETSEG.BAS on your HCJ Volume 2 diskette) provides one possible method. This program is written for use as a subroutine. In line 20030, the machine code is read from **DATA** statements and stored in the integer variable **ML%( )**. Once **ML%( )** is initialized, we use the statement **OFS% = VARPTR (ML%(0))** in line 20040 to get the offset of **ML%( )**. Now, line 20050 calls the machine code, and the DS register is returned in **DS%**. Because BASIC's default DS setting never changes, this routine only has to be called once.

It is important that *Get Seg* is stored within the BASIC workspace—that's why we store it in the BASIC variable **ML%( )**. If the machine code is stored outside the BASIC work area, calling it would require changing the DS register. If you change the DS register before calling *Get Seg,* the DS value returned would be that of the subroutine, not BASIC.

To see *Get Seg* put to practical use, refer to this volume's *PC Character Editor* program in IBM Focus. The program *PC Character Editor* stores its custom character set within the BASIC workspace in the variable **FONT%( )**. To "turn on" the custom characters, it's necessary to set a pointer to these new characters. This pointer requires both the offset and the DS of the new character set. With the *Get Seg* routine, we are able to find the DS of the custom characters and set the pointer accordingly.

File Verify

It's no fun when you lose an important disk file simply because you accidentally save a different file on top of it. This is, unfortunately, very easy to do. Many programs avoid this type of mistake by warning you when a file of the same name already exists on disk. If it exists, you are asked if you wish to replace the current file with the new one. Here, we offer two techniques for checking a disk for pre-existing files.

Both techniques presented here are subroutines written in Extended BASIC (see Listings 1 and 2 on this page). On your HCJ Volume 2 disk, these subroutines are saved as **MERGE** files under the file names VERIFY1 and VERIFY2. To load a **MERGE** file such as VERIFY1, enter the following: **MERGE DSK1.VERIFY1**

The first technique (see Listing 1) is the most versatile of the subroutines shown here. Not only can it check for data files (i.e., VAR 80), but it can also verify program files. To use this subroutine, set the variable **FL$** equal to the device & filename of the file that you want to verify and **GOSUB 10000**. The subroutine returns the variable **ST**; it will be set to a 1 if the file exists, or a 0 if the file does not. For instance, to verify the existence this very subroutine on your HCJ disk, **MERGE** VERIFY1 into memory and execute the following from within a program: **FL$="DSK1.VERIFY1" :: GOSUB 10000 :: PRINT ST :: END**

Assuming that your HCJ disk is in drive 1, variable **ST** should equal 1.

The way this subroutine works is simple. By **OPEN**ing a disk file without specifying a file name, we effectively open the disk's catalog. Now, we can read each file name in, one by one, until we find the file that we are looking for, or until we reach the end of the catalog. If the file is found, **ST** is set equal to 1. If the file is not found, **ST** is set equal to 0. Figure 1 lists the line annotations for this subroutine.

One problem with this aproach is error handling. If this subroutine encounters an error, such as reading an unformatted disk, or the absence of a disk in the drive, all succeeding disk access will fail. The initial error can be caught by the **ON ERROR** command, but any following disk access will cause an error before even spinning the disk. Because the catalog is so vital to a disk's existence, you must tread carefully while it is open.

The second file-verify technique (see Listing 2) is not as versatile as the first, but it does catch disk errors without complications. This subroutine can verify all types of files except program and **MERGE** files. You call this subroutine the same way as the first: Set the variable **FL$** equal to the device & filename of the file that you want to verify and **GOSUB 10000**. Again, upon returning, the subroutine returns the variable **ST** will be equal to a 1 if the file exists, or a 0 if the file does not.

This second subroutine uses a devious aproach to verifying files. To verify files, it *purposely* creates I/O errors. Basically, the file being verified is open for input. If an error occurs while the file is open, we know that the file does not exist. If an error does not occur, the file must exist. Figure 2 lists the line annotations for this subroutine.

Because of its error handling abilities, our HCJ Feature program, *FormFlex,* uses the second technique for verifying files. By using this subroutine, *FormFlex* can ask a user if he wishes to replace pre-existing files when saving his form data.

Figure 1

Line Nos.	
10000-10050	Program header
10060	Open disk's catalog
10070	Get disk's name
10080	Read file name
10090	If file name is the same, set ST and exit
10100	If not end of catalog then continue reading disk, otherwise clear ST and exit
10110	Close catalog
10120	Exit subroutine

Figure 2

Line Nos.	
10000-10050	Program header
10060	Initialize ST to 0 and set error trapping
10070	Open file for input If no error occurs, set ST to 1
10080	Exit subroutine
10090	Return from error to line 10080

Listing 1

```
10000 ! FILE VERIFY1
10010 ! VERIFIES ALL TYPES OF FILES, BUT HAS NO ERROR CHECKING
10020 ! COPYRIGHT 1986
10030 ! HOME COMPUTING JOURNAL
10040 ! VERSION 2.0
10050 ! TI EXTENDED BASIC
10060 OPEN #1:SEG$(FL$,1,5),INTERNAL,RELATIVE,INPUT
10070 INPUT #1:A$,A,A,A
10080 INPUT #1:A$,A,A,A
10090 IF A$=SEG$(FL$,6,20)THEN ST=1 :: GOTO 10110
10100 IF A$<>"" THEN 10080 ELSE ST=0
10110 CLOSE #1
10120 RETURN
```

Listing 2

```
10000 ! FILE VERIFY2
10010 ! VERIFIES ONLY DATA FILES, BUT HAS ERROR CHECKING
10020 ! COPYRIGHT 1986
10030 ! HOME COMPUTING JOURNAL
10040 ! VERSION 2.0
10050 ! TI EXTENDED BASIC
10060 ST=0 :: ON ERROR 10090
10070 OPEN #1:FL$,INPUT :: CLOSE #1 :: ON ERROR STOP :: ST=1
10080 RETURN
10090 RETURN 10080
```

Text On The Hi-Res Screen

Now you can display any of the Apple's characters on the hi-res screen with a simple PRINT command.

Note: *You can now make your own character shape tables and create custom characters. An editor program to help you do just that will be included in a later Volume of HCJ, so. . . stay tuned!*

One of the main reasons programmers shy away from the high resolution (hi-res) screen on the Apple is the lack of text support: Applesoft BASIC has made no provisions for displaying text on the hi-res screen. Of course, one could always use shape tables, or even **HPLOT** a character to the screen, but both of these methods are slow and cumbersome. Ideally, you want to be able to **PRINT** characters to the hi-res screen, just as if you were using the text screen. And to position the text, you want to use the **HTAB** and **VTAB** commands.

To remedy this situation, we have written a machine-language program called *Hi-Res Text*; with it, you can now output text to the hi-res screen with a simple **PRINT** statement from BASIC—without having to generate shape tables or calculate pixel locations.

There are three files on your HCJ Volume 2 disk that relate to the machine-language program presented here. These files are HIRESTEXT.BIN, HIRESTEXT, and NOTEPAD. The first file, HIRESTEXT.BIN, is the actual machine code for the *Hi-Res Text* program. The HIRESTEXT file is a BASIC program that installs *Hi-Res Text* by moving the BASIC workspace above the hi-res screen, **BLOAD**ing HIRESTEXT.BIN at memory location 2048 ($0800), and then running the BASIC program of your choice. Currently, HIRESTEXT is set up to run NOTEPAD, the third file mentioned here. *Notepad* is a sample program that takes advantage of the text capabilities that *Hi-Res Text* offers. The *Notepad* program is listed on the following page.

Using *Hi-Res Text*

The first step to using *Hi-Res Text* is changing line 220 in the BASIC program HIRESTEXT to run *your* BASIC program instead of *Notepad*. To make the program run a file called PIECHART, for example, change line 220 to look like this:

```
220 PRINT CHR$(4);"RUN PIECHART"
```

You may also want to rename the BASIC program HIRESTEXT to a more appropriate name, such as BOOT.PIE. If you rename the program, however, you must also change line 200 to reflect the change.

Now, to install *Hi-Res Text* from within your program, execute a **CALL 2808**. To de-activate *Hi-Res Text*, execute a **CALL 2811**. When *Hi-Res Text* is activated, all **PRINT** statements send text to both the text screen and the hi-res screen. When de-activated, **PRINT** statements send text only to the text screen.

Our sample program *Notepad* provides a prime example of how to use *Hi-Res Text* to enhance your BASIC programs (see listing).

Using *Notepad*

To use this program, simply **RUN** HIRESTEXT. *Notepad* will then automatically load and run for you.

The sample program, *Notepad*, simulates a real notepad for you to write, print, and save notes. This program was inspired by the Notepad desk accessory available on the Apple Macintosh.

To draw a graphic representation of a notepad, we used the hi-res screen. To display text entered on the notepad, we used our program *Hi-Res Text*.

The notepad has 8 pages. To move from one page to another, hold down the **Open** ♦ key, and press any number **1** through **8** (corresponding to the page that you wish to turn to). To enter text on the notepad, simply type. To edit your typing, you can backspace with the **[Delete]** key and correct your mistakes. If you wish to erase the entire page, press **Open** ♦ **C**. To print the page, press **Open** ♦ **P**. The **[Esc]** key exits the program.

When *Notepad* is first run, it attempts to load the text file, NOTEPAD.TXT. If this file is not present, the notepad comes up blank. Whenever you exit *Notepad*, the program attempts to create the file NOTEPAD.TXT. If a non-write-protected disk is in the drive, your notepad will be preserved for later editing. You can have one notepad per disk. Just remember to copy the files HIRESTEXT.BIN, HIRESTEXT, and NOTEPAD onto every disk that you want your notepad on. Remember, to use *Notepad*, you must first run the BASIC file HIRESTEXT.

CONTROL CAPSULE

Notepad

KEY	FUNCTION
Delete	Backspace over text
Open ♦ 1 through 8	Turn to selected page
Open ♦ P	Print the page
Open ♦ C	Clear the page
Esc	Exit program

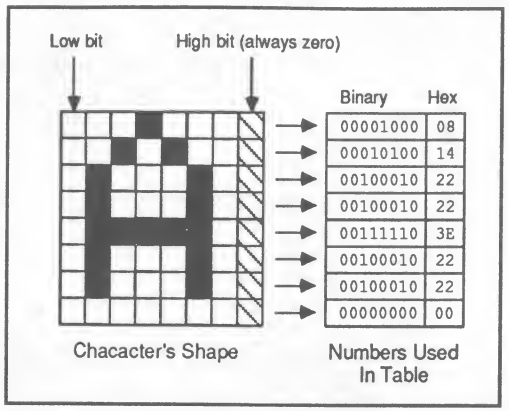


Figure 1.

Each of the character's pixels are represented by a binary digit (a bit). So, each row in a character's shape is converted into one byte. The leftmost pixel in each row, however, is considered the low bit of the resulting byte, not the high bit. Because of this, the final binary representation of a character appears as a mirror image of the actual character's shape.

How It Works

When you execute a **CALL 2808**, the character output vector located at 48688 ($BE30) is changed to point to our character graphics routine. A **CALL 2011**, restores this vector to its original value.

Once the character output vector has been changed, we can intercept all **PRINT** statements. So now, when a character is output, we use a table to find the shape of the character, get the hi-res screen's location in memory, and "print" the character to the hi-res screen.

The real key to this program is the program's own character shape table. This table contains 8 bytes of shape information for every printable character (ASCII 32 through ASCII 126). Using a character's ASCII value, we can index into the table and find its shape. And, because we use the same data format that the Apple uses to store graphic images on the hi-res screen, our routine is quicker and more memory conserving than conventional **DRAW** and **XDRAW** shape tables. The format used by this program for its character shape table is illustrated in Figure 1. The actual table is located in memory at 2048-2807 ($0800-$0AF7).

The character shape table that the program uses is part of the object file HIRESTEXT.BIN. Because this table loads into RAM, it is possible to modify the characters' shapes after HRESTEXT.BIN has been loaded. By modifying the character shape table, you can create you own custom character sets for use on the hi-res screen. This, however, would require another program.

Listing 1.

```
100  REM ***********
110  REM * NOTEPAD *
120  REM ***********
130  REM COPYRIGHT 1986
140  REM HOME COMPUTING JOURNAL
150  REM VERSION 2.0
160  REM  APPLE II FAMILY APPLESOFT
170  REM
180 P1 = 2808:P0 = 2811
190 NX = 8:NY = 2
200 PG = 1
210 MP = 8:MX = 16:MY = 14
220  DIM NP$(MP)
230  CALL P1
240  REM
250  REM  GET NOTEPAD
260  REM
270  ONERR  GOTO 860
280  PRINT  CHR$ (4);"VERIFY NOTEPAD.TXT"
290  PRINT  CHR$ (4);"OPEN NOTEPAD.TXT"
300  PRINT  CHR$ (4);"READ NOTEPAD.TXT"
310  FOR IT = 1 TO MP
320  GET KP$: IF KP$ <  >  CHR$ (127) THEN NP$(IT) =
NP$(IT) + KP$: GOTO 320
330  NEXT IT
340  PRINT  CHR$ (4);"CLOSE NOTEPAD.TXT"
350  POKE 16,0
360  REM
370  REM DRAW NOTEPAD
380  REM
390  HOME : HGR : HCOLOR= 3
400 X1 = NX * 7 - 1:Y1 = NY * 8 - 1:X2 = X1 + MX * 7
+ 1:Y2 = Y1 + (MY + 2) * 8 + 1
410  HPLOT X1,Y1 TO X2,Y1 TO X2,Y2 + 2 TO X1,Y2 + 2 TO
X1,Y1
420  HPLOT X1,Y1 - 8 TO X2,Y1 - 8 TO X2,Y1 TO X1,Y1 TO
X1,Y1 - 8
430  FOR IT = Y1 - 8 TO Y1 STEP 2: HPLOT X1,IT TO X2,IT:
NEXT
440  HPLOT X1 + 1,Y2 + 3 TO X2 + 1,Y2 + 3 TO X2,Y1 - 8
450  HPLOT X1 + 16,Y2 TO X1,Y2 - 16 TO X1 + 16,Y2 - 16
TO X1 + 16,Y2 TO X2,Y2
460  POKE 32,NX: POKE 33,MX: POKE 34,NY: POKE 35,NY + MY
470  HOME : FOR IT = 1 TO MY:: PRINT  LEFT$ ("
                                ",MX);: NEXT IT
480  HTAB  INT (MX / 2) - 1: VTAB NY + MY + 2: PRINT "-
";PG;"-";
490  HOME : PRINT NP$(PG);
500  REM
510  REM MAIN SERVICE LOOP
520  REM
530 X =  PEEK (36) + 1:Y =  PEEK (37) + 1: HTAB X:
VTAB Y: PRINT "_";: HTAB X: VTAB Y: GET KP$:OA =
PEEK ( - 16287):KP =  ASC (KP$)
540  IF OA > 127 THEN 630
550  IF KP = 27 THEN 700
560  IF KP <  > 127 OR (X = 1 AND Y = NY + 1) THEN 590
570  PRINT " ":L =  LEN (NP$(PG)) - 1: IF L = 0 THEN
NP$(PG) = "": GOTO 490
580 NP$(PG) =  LEFT$ (NP$(PG),L): GOTO 490
590  IF KP = 13 AND Y <  > MY + NY THEN NP$(PG) =
NP$(PG) + KP$: PRINT " ";: GOTO 490
600  IF X = MX AND Y = MY + NY THEN  PRINT  CHR$ (7);:
GOTO 530
610  IF KP > 31 AND KP < 127 THEN  PRINT KP$;:NP$(PG) =
NP$(PG) + KP$: GOTO 530
620  GOTO 530
630  REM
640  REM  OPEN APPLE COMMANDS
650  REM
660  IF KP > 48 AND KP < 57 THEN PG = KP - 48: GOTO 470
670  IF KP = 112 OR KP = 80 THEN  CALL P0: PRINT  CHR$
(4);"PR#1": PRINT : PRINT "PAGE ";PG: PRINT NP$(PG):
PRINT  CHR$ (4);"PR#0":
CALL P1: GOTO 480
680  IF KP = 67 OR KP = 99 THEN NP$(PG) = "": GOTO 470
690  GOTO 530
700  REM
710  REM PUT NOTEPAD AWAY
720  REM
730  CALL P0
740  ONERR  GOTO 870
750  PRINT  CHR$ (4);"OPEN NOTEPAD.TXT"
760  PRINT  CHR$ (4);"CLOSE NOTEPAD.TXT"
770  PRINT  CHR$ (4);"DELETE NOTEPAD.TXT"
780  PRINT  CHR$ (4);"OPEN NOTEPAD.TXT"
790  PRINT  CHR$ (4);"WRITE NOTEPAD.TXT"
800  FOR IT = 1 TO MP: PRINT NP$(IT); CHR$ (127);: NEXT
IT
810  PRINT  CHR$ (4);"CLOSE NOTEPAD.TXT"
820  TEXT : HOME : END
830  REM
840  REM ERROR ROUTINES
850  REM
860  CALL  - 3288: GOTO 350
870  CALL  - 3288: GOTO 820
```

Pull-Down Menus

Order up some pull-down menus for your Atari computer—the perfect choice for the byte-conscious gourmet.

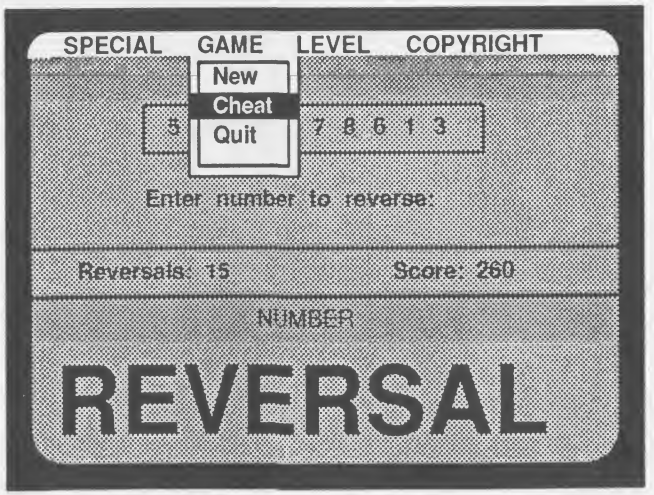
More and more computer software is using pull-down menus to simplify the interface between users and computers. Computers such as the Atari ST, Commodore Amiga, and Apple Macintosh have made pull-down menus an integral part of their operating systems. Although pull-down menus are not a standard feature on the Atari 800 family of computers, they can be added with relative ease—that is, when you use the accompanying program, *Atari Menus*.

Atari Menus is a package of subroutines that add pull-down menus to any BASIC program. With these subroutines, you can define menus and their options, display the menu bar (a list of available menus), and most importantly, allow the user to pull down (display) a menu and select an option. *Atari Menus* is supplied on your HCJ Volume 2 disk under the file name of MENUS.TXT. This is a text file that can be merged into memory with the following command: **ENTER "D:MENUS.TXT"**

Oh Waiter, Could I Have A Menu Please?

To define your menus, you must enter certain information into **DATA** statements. Look at lines 30290 to 30330 of *Atari Menus* to see an example. Here we have entered some sample data. We suggest that you refer to these **DATA** statements while reading the explanation of their format.

This screen simulation shows how pull-down menus look in our demonstration game—Number Reversal.



The first piece of data specifies the number of menus that will appear on the menu bar. The data that follows provides the actual menu information. This data consists of the menu name, the number of options in the menu, and then the option names. This information repeats for each menu. Figure 1 summarizes the menu data format.

With the **DATA** statements all set up, you're ready to install pull-down menus in a program. There are three main subroutines in *Atari Menus*. These subroutines should be called in the order in which they are listed here. The first subroutine (**GOSUB 30010**) initializes the pull-down menus by reading the menu data and setting up some key variables. You must ensure that the computer's **DATA** pointer is **RESTORE**d to your menu data when you call this subroutine.

The second subroutine (**GOSUB 30350**) simply displays the menu bar at the top of the screen.

The third subroutine (**GOSUB 30740**) operates the pull-down menus. When called, the user is allowed to pull down any of the menus and select an option. There are two variables that are passed back from this last routine—the variables **MENU** and **OPT**. **MENU** is set equal to the number of the menu from which a selection was made. This number corresponds to the position of the menu on the menu bar. **OPT** is set equal to the number of the option that was selected. This number corresponds to the position of the option within the menu. If the user did not select an option, **OPT** will equal zero. See Figure 2 for a list of these **GOSUB**s and their function.

I'm Ready To Order Now

Making selections from the menu is easy. When the menus are made operative (via a **GOSUB 30740**), the first menu on the menu bar is automatically pulled down and made active. To select an option, use the cursor-up and cursor-down keys. To move from one menu to the next, use the cursor-left and cursor-right keys. As you move from menu to menu, the old menu will close (push up?) as the new one is pulled down. Note that any screen information that is covered up by a menu is restored when the menu is closed. Pressing **[RETURN]** makes your menu selection final. If you decide not to choose any of the options, press **[ESC]**.

Because of the nature of pull-down menus, the joystick-controlled cursor program in this issue's Atari Technote provides a perfect input device. If you modify the routine so that the joystick's fire button simulates the **[RETURN]** key, as described in the Tech Note article, you can sit back and make menu selections without ever touching the keyboard.

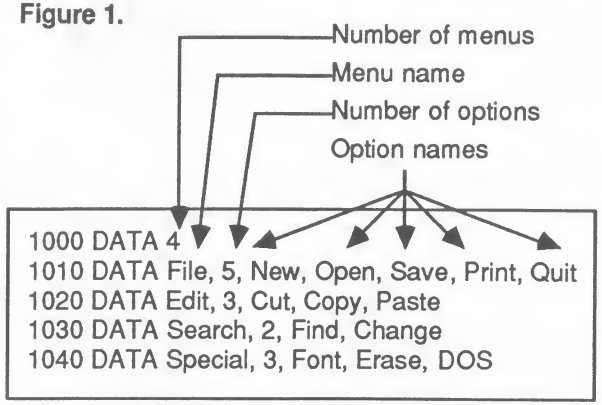
Considering The Menu's Right-Hand Column

As with most things in life, there is a price you have to pay when using *Atari Menus*. Fortunately, the price is small—only about 4K of memory. *Atari Menu's* code and variables take up approximately 4K of the BASIC workspace (more or less depending on the number and size of menus defined). Not a major chunk, but something to consider when using *Atari Menus* within a large program.

Atari Menus uses **GRAPHICS** mode zero. This is the 40-column text screen. Please take into account the size of the screen when defining your pull-down menus. If you define too many menus, or use names that are too long to fit on the screen, *Atari Menus* will not work properly.

Atari Menus uses 20 variables (see Key Variables). Avoid using variables of the same name in your host program. One more thing to consider: *Atari Menus* is contained in line numbers 30000 to 30860. If your program uses these line numbers, you must re-number *Atari Menus*.

Figure 1.



Today's Blue-Plate Special

On your HCJ Volume 2 disk is a program called *Number Reversal.* It is saved under the file name of REVERSAL; the program's Control Capsule appears on this page. This program takes advantage of the pull-down menus provided by *Atari Menus.* It is a good example of how to use *Atari Menus* in a program. Line 240 initializes the menus, line 470 displays the menu bar, and line 650 activates the menus whenever **[CTRL] M** is pressed.

Number Reversal is a game of logic in which you are presented with a series of digits. You goal is to rearrange the digits into numeric order. This may seem like a menial undertaking at first, but wait until you play the game. . . To reorder the numbers you must *reverse* the order of selected digits. To choose the number of digits to reverse, select one of the digits. Once selected, all digits to the left of, and including the selected digit are reversed. Sound confusing? Don't worry—it's supposed to be! Just run the program and experiment. If you're persistent, you'll get it.

Figure 2.

Subroutine	Function
GOSUB 30010	Initialize menu routines
GOSUB 30350	Display menu bar
GOSUB 30740	Operate pull-down menus

KEY VARIABLES

Atari Menus

Variable	Function
MAXMENUS	Number of menus
NUMOPTS( )	Number of options for each menu
MENUBAR$	String of menu names
OPTIONS$	String of menu option names
MENUX( )	Horizontal position of each menu
OPTIND( )	Index into OPTION$ for each menu
MENULEN( )	Length of each menu name
OPTLEN( )	Length of option names for each menu
SCREEN$	String to hold screen data
SCREEN	Address of text screen
MENU	Menu from which selection was made
OPT	Option that was selected
REV$	ML routine to reverse characters
LINE$	Multi-purpose string variable
MX	Multi-purpose numeric variable
MI, MJ	Loop counters
MM	Current menu showing
MO	Current option highlighted
MK	Keypress

CONTROL CAPSULE

Number Reversal

KEY	FUNCTION
0 through 9	Select digit to reverse
CTRL M	Use pull-down menus

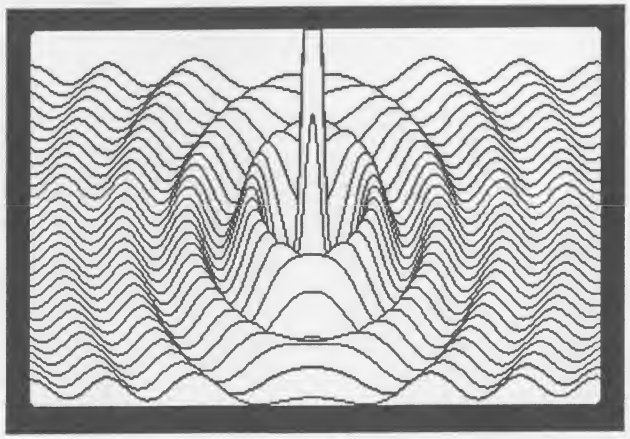
Using Pull-Down Menus

Cursor Left	Move to the menu on the left
Cursor Right	Move to the menu on the right
Cursor Up	Select previous option
Cursor Down	Select next option
RETURN	Make menu selection final
ESC	Exit pull-down menus without making selection

Get out your 3D glasses, because the C-64 is crossing the dimension barrier.

One of the reasons that the Commodore 64 computer is so popular is because of its many different graphics modes: There's standard character mode, extended-color character mode, multi-color character mode, standard bit-map mode, and multi-color bit-map mode. All of these modes are capable of producing impressive graphic displays, but not all of them are easy to use—specifically bit-map mode. From BASIC, bit-mapped graphics are cumbersome to use, difficult to program, and—last but not least—*slow*. To simply draw a diagonal line across the screen can take up to a minute.

To remedy this situation, we have put together a package of high-resolution graphics routines for standard bit-map mode. These routines are written in machine language and are accessible from both BASIC and machine language. We call this package *HRG-64*.



The Basics of Bit-Map

First, an explanation of bit-mapped graphics is in order. In character mode, you have a grid of 40 by 25 character positions. This works out to be 1000 (40*25) different locations in which you can place one of many characters. Now, if you have ever defined your own character set, you have learned that each character consists of an even finer grid of 8 by 8 dots. These dots (called pixels) can be either on or off. A pixel that is "on" displays the foreground color; a pixel that is "off" displays the background color. It is these pixels that make up a character's shape.

Taking this all into account, we can conclude that the Commodore 64's screen is actually a grid of 320 by 200 pixels. You see, if there are 40 by 25 character positions, each holding an 8-by-8 grid of pixels, that gives the screen a total of 320 (40*8) by 200 (25*8) pixels—64000 pixels in all! In character mode, you must display and/or redefine characters in order to control these pixels. Bit-mapped graphics provides complete control over each individual pixel, instead of being limited by character definitions.

As mentioned above, there are two bit-map modes—standard and multi-color. The differences between the two has to do with resolution and color. In standard mode, you have the complete 320 by 200 pixel grid to work with. A pixel can display either the foreground or background color. Also, the foreground and background color can be different for every character (8 by 8 pixel) location. In muti-color mode, your horizontal resolution is cut in half. Instead of a 320 by 200 pixel grid, your are left with a 160 by 200 pixel grid. Because of this, the pixels appear twice as wide on the screen. With the reduced resolution, however, comes greater color flexibility. A pixel can now display one of three foreground colors or the background color. The background color remains the same for the whole screen, but the three foreground colors can be different for every character location. Because of the superior resolution, *HRG-64* works in standard bit-map mode.

Figure 1.

Register	Memory Location (in decimal)
A	780
X	781
Y	782
P (Status)	783

Using *HRG-64*

To use *HRG-64*, you must first load it into the computer's memory. This is done by placing your HCJ disk for Volume 2 into your disk drive and entering **LOAD "HRG-64",8,1**. When the cursor returns, enter **NEW**, and you're ready to go.

HRG-64 consists of 7 routines. These routines allow you to turn on and off the graphics screen, clear the graphics screen, set the foreground and background color, plot points, and draw lines. To use any of these routines, you must first load the 6510's A register with the number (0 through 6) of the routine that you wish to call, and then SYS (from BASIC) or JSR (from machine language) to *HRG-64's* entrance point located at 51264 ($C840). Now, some of you BASIC programmers may be a little worried about that statement, "load the 6510's A register." Well, don't be. This is very easily done from BASIC. All of the 6510s registers can be set from BASIC prior to a SYS statement by POKEing locations 780-783. See Figure 1 for a list of the different registers and the corresponding memory locations used by the SYS command.

So, to call the first of *HRG-64's* routines from BASIC, you would use the following code: **POKE 780,0:SYS 51264.** The machine code equivalent is:

```
LDA #0
JSR 51264
```

This example, by the way, turns on the graphics screen by putting the Commodore 64 into bit-mapped graphics mode. Whenever you call a *HRG-64* routine, none of the 6510's registers are effected—they will remain the same as when the routine was first called. This is a convenient feature when calling these routines from machine language.

Chart 1 lists all seven of the high-resolution graphics routines and provides full explanations on how to use them. Note that all parameters are passed to these routines through the 6510's registers.

A Three-Dimensional Example

To show off *HRG-64's* capabilities, we provide a sample program on your HCJ diskette. This program is called *Ripples*. *Ripples* has been a long time favorite of our editorial staff. For more information on the physics and programming behind *Ripples*, see Atari Focus in HCJ Volume 1. *Ripples* displays a ripple pattern on the computer screen in stunning three-

dimensional graphics. This particular version provides four different ripple patterns: ripples *without* a splash, hyperbolic spike, ripples *with* a splash, and square ripples with a splash. When *Ripples* is RUN, you can choose from these four patterns. After the screen has been drawn, pressing a key will return you to the program's menu. Because creating the ripple patterns requires several complex calculations, completing the entire screen takes a fair amount of time. But, when benchmarked against other computers (i.e., Apple, Atari, and IBM), the Commodore 64 finished first!

Programming Considerations

If you are a machine language programmer, there is some additional information that you may require. First off, *HRG-64* is stored in memory at 51200 ($C800) to 52207 ($CBEF). When the high-resolution screen is in use, memory locations 52224 ($CC00) to 53247 ($CFFF) are used for the bit-mapped screen's color memory. This means that the 2K chunk of memory, 51200 ($C800) to 53247 ($CFFF), is no longer available for machine code. The actual bit-mapped screen, however, is neatly tucked away underneath kernal ROM located at 57344 ($E000) to 65535 ($FFFF), where it should bother no one. "What's left?" you may ask. Well, you still have use of the cassette buffer located at 828 ($033C) to 1019 ($03FB) and the bottom 2K of free RAM located at 49152 ($C000) to 51199 ($C7FF).

If you are planning to use sprites in conjunction with the high-resolution screen, there are a couple things that you must consider. *HRG-64* uses bank 3 for all video information. Because of this, all sprite data must be stored somewhere in memory between 49152 ($C000) to 65535 ($FFFF). Sprite pointers are now located at 53240 ($CFF8) to 53247 ($CFFF).

Chart 1.

Turn On Graphics Screen:
Register **A** = 0

Param.	Value	Function
X	0 or <>0	Don't clear screen or clear screen

Remarks: This turns on the graphics screen. If the **X** register is zero, the graphics screen will not be cleared. If the **X** register does not equal zero, the graphics screen will be cleared.

Example: Turning on and clearing the bit-mapped graphics screen

BASIC	Machine Code
POKE 780,0	LDA #0
POKE 781,1	LDX #1
SYS 51264	JSR 51264

Turn Off Graphics Screen:
Register **A** = 1

Param.	Value	Function
-none-		

Remarks: This turns off the graphics screen by returning you to standard character mode.

Example: Turning off the graphics screen

BASIC	Machine Code
POKE 780,1	LDA #1
SYS 51264	JSR 51264

Clear Graphics Screen:
Register **A** = 2

Param.	Value	Function
X	0-255	Set bit pattern with which to clear screen

Remarks: This routine clears the screen using whatever bit pattern is placed in the **X** register. To clear the screen to the background color, place a 0 in the **X** register. To clear the screen to the foreground color, place a 255 in the **X** register. Most other numbers result in a striped-looking screen.

Example: Clearing the screen to the background color

BASIC	Machine Code
POKE 780,2	LDA #2
POKE 781,0	LDX #0
SYS 51264	JSR 51264

Set Colors:
Register **A** = 3

Param.	Value	Function
X	0 or 1	Set background or foreground color
Y	0-15	Color

Remarks: This routine lets you define the screen's background and foreground colors. If **X** is set to 0, you will be setting the background color. If **X** is set to 1, you will be setting the foreground color. The **Y** register specifies the color (0-15) that you want. The default background color is black (0). The default foreground color is white (1).

Example: Setting the foreground color to yellow

BASIC	Machine Code
POKE 780,3	LDA #3
POKE 781,1	LDX #1
POKE 782,7	LDY #7
SYS 51264	JSR 51264

Set Pen to Draw or Erase:
Register **A** = 4

Param.	Value	Function
X	0 or <>0	Set pen to erase or draw

Remarks: This routine defines the outcome of when you plot a point or draw a line. If the **X** register is set to zero when you call this routine, all Plot and Drawto commands will plot points using the background color (thus *erasing* points on the screen). If the **X** register is set to a non-zero value, all Plot and Drawto commands will plot points using the foreground color (thus *drawing* points on the screen).

Example: Setting pen to erase

BASIC	Machine Code
POKE 780,4	LDA #4
POKE 781,0	LDX #0
SYS 51264	JSR 51264

Plot:
Register **A** = 5

Param.	Value	Function
X	0-255	Low byte of X coordinate
P	0-1	High byte of X coordinate
Y	0-199	Y coordinate

Remarks: This routine allows you to plot points on the high-resolution screen. Coordinates (0,0) are the upper left corner of the screen. Note that the X coordinate must be between 0 and 319 and the Y coordinate must be between 0 and 199. Values other than these *will* cause errors. Because the X coordinate can be as great as 319 —with a register's maximum value being 255—the **P** register is used to hold the high byte of the X coordinate. From machine language, all you have to do is set the carry flag (SEC) when the X coordinate is greater than 255. This produces the same result as setting the **P** register equal to 1.

Example: Plotting a point at coordinates (300,150)

BASIC	Machine Code
POKE 780,5	LDA #5
POKE 781,44	LDX #44
POKE 783,1	SEC
POKE 782,150	LDY #150
SYS 51264	JSR 51264

Drawto:
Register **A** = 6

Param.	Value	Function
X	0-255	Low byte of X coordinate to Drawto
P	0-1	High byte of X coordinate to Drawto
Y	0-199	Y coordinate to Drawto

Remarks: This routine allows you to draw lines on the high-resolution screen. The computer will draw a line from the last point plotted to the point specified by the **X**, **P**, and **Y** registers. If no previous point has been plotted, the computer will start from the upper-left corner of the screen. Note that the X coordinate must be between 0 and 319 and the Y coordinate must be between 0 and 199. Values other than these *will* cause errors. Because the X coordinate can be as great as 319—with a register's maximum value being 255—the **P** register is used to hold the high byte of the X coordinate. From machine language, all you have to do is clear the carry flag (CLC) when the X coordinate is less than 255. This produces the same result as setting the **P** register to equal 0.

Example: Drawing a line to coordinates (10,10)

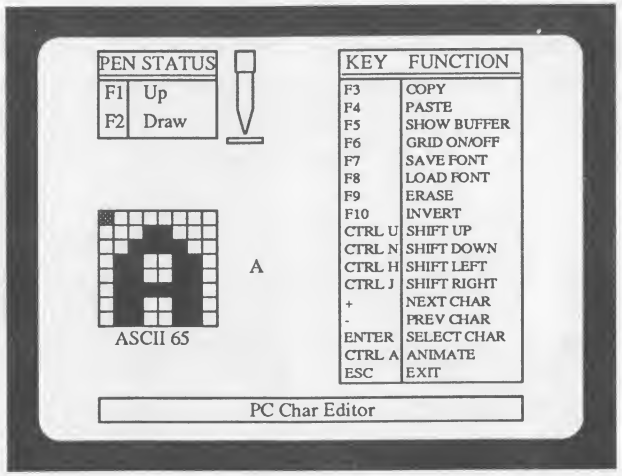
BASIC	Machine Code
POKE 780,6	LDA #6
POKE 781,10	LDX #10
POKE 783,0	CLC
POKE 782,10	LDY #10
SYS 51264	JSR 51264

PC Character Editor

Adding character to your BASIC programs.

One of the most overlooked capabilities of the PC is character redefinition. With a Tandy 1000, PCjr, or IBM PC with Color/Graphics Adapter, it is possible to create your own custom character sets. Besides designing your own fonts and foreign characters, custom characters provide a simple means of producing graphics and animation.

The program presented here, *PC Character Editor*, allows you to design your own character sets. You can load and save these character definitions for later editing or use in other programs. Special features include inverting a character's image, shifting its shape in any direction, and rapidly flipping through a sequence of characters for an animated effect.



CONTROL CAPSULE

PC Character Editor

KEY	FUNCTION	KEY	FUNCTION
Cursor keys	Move cursor	f10	Invert character
f1	Pen Up/Down	Ctrl U	Shift character up
f2	Pen Draw/Erase	Ctrl N	Shift character down
f3	Copy character	Ctrl H	Shift character left
f4	Paste character	Ctrl J	Shift character right
f5	Show buffer	+	Next character
f6	Grid on/off	-	Previous character
f7	Save font	Enter	Select character to go to
f8	Load font	Ctrl A	Animate character
f9	Erase character	Esc	Exit program

On your HCJ Volume 2 disk, you will find two different versions of *PC Character Editor*. The first version is written in BASIC, and is saved as CHAREDIT.BAS on this disk.

The second version is compiled BASIC. This version is compiled with Microsoft's QuickBASIC Compiler 2.0. The compiled version is executable from DOS—simply enter **CHAREDIT** from the A> prompt (assuming that the HCJ Volume 2 disk is in drive A:). This version does not require BASIC, but it does require at least 256K of memory to be run. If possible, we advise that you use the compiled version whenever running *PC Character Editor*.

Certain Limitations

Before describing how *PC Character Editor* works, there are a few limitations you should be aware of. First off, you can only display the modified character set on a graphics screen (any screen other than **SCREEN 0**). Secondly, if you are using an IBM PC, you can only re-define the upper-level ASCII characters (**CHR$(128)** to **CHR$(255)**). Attempting to re-define any of the low-level ASCII characters on the IBM PC has no effect.

On all of the PC-compatible machines there are certain characters that cannot be displayed. These include such characters as **CHR$(7)** (the bell), or **CHR$(12)** (clear screen). Although these characters have shapes associated with them, they cannot be displayed from BASIC. In these cases, *PC Character Editor* will display the character as a space.

Using The Editor

When you run *PC Character Editor*, you are presented with the editing screen (see the accompanying screen simulation). All program options are accessed via a keypress from this screen.

In the middle-left part of the screen is your design pad. The design pad is an 8 X 8 grid, representing an enlarged character. Inside the design pad is your cursor, flashing away; use your computer's cursor keys to move it. You'll be using the cursor to design characters (more on this later).

Directly below the design pad is the ASCII value of the character that you are currently editing. To the right of the design pad is the actual character. Whenever you modify something in the design pad, the character on the right is updated.

Above the design pad is the pen status. The pen status determines how your cursor affects the character being edited in the design pad. The pen can be set to Draw or Erase; it can also be Up or Down. If the pen is Up, moving the cursor has no effect on the character being edited. When the pen is Down, moving the cursor either draws or erases pixels (depending on whether the pen is set to Draw or Erase).

On the right, is an on-screen Control Capsule. All of *PC Character Editor's* keypresses are documented here. Although most of *PC Character Editor's* functions are self-explanatory, some require further instruction.

First, moving to the previous or next character is accomplished easily by pressing the - or + keys. To jump to a selected character, you must press **[Enter]**. Now, to select a character, press the key that corresponds to the character that you wish to edit. (To move to **CHR$(65)**, for example, press the **A** key.) Certain characters, however, do not have a corresponding key (i.e., **CHR(128)**). In this case, you must use a keyboard feature that is unique to the IBM PC and compatibles: By holding down the **[Alt]** key, typing in the desired character's three-digit ASCII number on the numeric keypad, and then releasing the **[Alt]** key, you can enter any of the 256 ASCII codes. If you own a PCjr, you must first activate your numeric keypad by holding down the **[Alt]** key, and then pressing **[Fn] N**. Now, the upper row of number keys act just like the PC's numeric keypad.

PC Character Editor's most interesting feature is its Animate function. Animation is generally the result of several similar, but different pictures being displayed in rapid succession. This is essentially the basis behind *PC Character Editor's* Animate function—only instead of pictures, we display characters.

When you activate the Animate function by pressing **[Ctrl] A**, you are prompted for a starting character and an ending character. To enter these characters, use the same method described above for selecting an edit character. Once selected, the Animate routine repeatedly cycles between the starting and ending characters. To speed up this sequence, press the cursor-

up key. Conversely, the cursor-down key slows down the sequence. Pressing [Esc] exits Animate mode.

For the duplication of character shapes, we offer the Copy and Paste options. When you select Copy by pressing **f3**, the shape of the current character is placed into a buffer. The buffer contents can be displayed by pressing **f5**. Now, whenever you select Paste by pressing **f4**, the current character's shape is replaced by the character in the buffer. Duplicating a character's shape comes in very handy when creating animated sequences.

You may also load and save your character sets. A character set comprises all 256 ASCII characters. When you choose one of these options, the computer prompts you for a file name and then saves or loads your file. Character set files are saved with the three character extender FNT, and take up approximately 2K of disk space.

Building Character

No one likes the "blank-paper blues," so to get you started with some pre-built characters, we have provided two character set files on your HCJ Volume 2 disk. These character sets contain the standard characters found on the IBM PC and Tandy 1000. The first file is saved as IBMFONT and the second as TANDFONT. To load these files, select the Load Font option, insert your HCJ disk, and enter one of the above file names.

How It Works

Down in the lower section of memory are two four-byte pointers that tell the computer where to find its character set's image table. One of these pointers is for the upper-level ASCII characters, and the other is for the lower-level ASCII characters. With the data segment register set at &H0000, the upper-level ASCII pointer is located at &H0110 and the lower-level ASCII pointer is at &H007C. The first two bytes of these pointers contain the segment address of the character set's image table. The following two bytes contain the offset. By multiplying the segment address by 16, and then adding the offset, you can find the absolute address of the character set's image table in memory.

These pointers normally point to an area in ROM, but *PC Character Editor* changes the pointers to point at its own character image table located in RAM. To store the image table, *PC Character Editor* uses the integer array **FONT%()**. This array is **DIM**ed to 1023 elements, providing 2K of storage space.

Adding Character To Your Programs

We have assembled a series of subroutines that allow you to use custom characters in your own programs. These subroutines are saved as CHARSUBS.BAS on your HCJ Volume 2 disk. This is a text file that can be **LOAD**ed or **MERGE**d into your BASIC programs. Use of these subroutines require that you **DIM**ension three variables within your program. This requires the following code: **DIM FONT%(1023),ML%(6),OLDVEC(1,3)**

FONT%() stores the custom character set, **ML%()** stores the machine code to get BASIC's data segment (see this Volume's IBM Technote), and **OLDVEC(,)** stores the old character set pointer, so that we can restore the original character set when exiting the program.

These subroutines provide you with the means to load and save character sets generated by *PC Character Editor*, install the custom characters, and restore the old character set. Before any of these subroutines can be used, however, you must execute a **GOSUB 10180:GOSUB 10250** from within your program. The first **GOSUB** gets BASIC's data segment; the second saves the original character set pointers. The various subroutines, and their functions are expained in Figure 1.

A couple of warnings about using custom characters in a program: First, never exit a program without restoring the characters back to their original shape. If you have changed a character so that it looks like another, the computer will confuse the two, and consider both characters to be the same. The character with the lowest ASCII gets chosen. In other words, if you make a B look like an A, then every time you type B, the computer will consider it an A instead.

Because the new character set is stored in a BASIC variable, it is possible for it to get relocated in memory. So, before you turn on the new character set, be careful to predefine any variables that may be used while custom characters are in use.

Two prime examples of using custom characters in a BASIC program are *PC Character Editor*, and the mini-game *Nukeman*, provided on your HCJ disk. The accompanying sidebar describes how the game is played.

Figure 1.

Subroutine	Function
GOSUB 10010	Turn on custom characters
GOSUB 10110	Turn off custom characters (restore original characters)
GOSUB 10180	Get original character vector
GOSUB 10250	Get BASIC's data segment register
GOSUB 10330	Load a character set created by *PC Character Editor*
GOSUB 10390	Save a character set

*Note that your program should execute the following two lines before any of these routines can be used:

```
DIM FONT%(1023),ML%(6),OLDVEC(1,3):GOSUB 10180:GOSUB 10250
```

Nukeman

It's a bird, it's a plane, it's a Trident missle . . . No, it's Nukeman!

Due to a freak laboratory explosion, mild-mannered nuclear physicist Luke F. Huntley was exposed to dangerously high levels of radiation. To the amazement of his colleagues, Luke managed to escape the demolished research center with his life (and his lunch, which was in the lab's refrigerator). Luke, however, did not escape a normal man; he had become something else, something strange, something bizarre, something without much hair... Luke had become *Nukeman*.

In the months to follow, the government brought Luke to a small town in Nevada, and ran him through several extensive tests: They tested his reflexes; they took blood samples; they injected him with drugs; and they dropped nuclear bombs on him. It was during this time that the government came to realize the supernatural power imposed upon Luke by his laboratory mishap—he was impervious to nuclear weapons.

Now, it just so happens that about this time, a young boy armed with a personal computer, a telephone modem, and an insatiable desire to break into computer systems, began World War III. This was not taken lightly by the government.

Well, the boy was caught, but intelligence now reports 20 nuclear warheads heading toward American soil... and there's only one man who can stop them—Nukeman. If Nukeman can throw his body beneath the oncomming bombs before they hit the ground, he can absorb the nuclear blast and save the nation.

So, your mission—if you choose to accept it—is to manuever Nukeman under each descending bomb, thus preventing the destruction of the free world, as we know it today...

CONTROL CAPSULE

Nukeman

KEY	FUNCTION
Left Shift	Move Nukeman left
Right Shift	Move Nukeman right

TI-Writer, look out!
Here comes HCJWord. . .

Need to jot down a few notes, write a letter to your aunt, or crank out a quick resume before a job interview? *HCJWord* may be just what you're looking for. *HCJWord* is an easy-to-use word processor that allows you to edit, save, load, and print a page of text quickly and painlessly. *HCJWord* requires Extended BASIC and 32K memory expansion. (It also helps to have a printer.)

To boot *HCJWord,* simply **OLD** the file **HCJWORD** from your HCJ Volume 2 disk and enter **RUN**. You may also select *HCJWord* from the *HCJ Director* program.

A Page Processor

HCJWord is what you might call a "page processor." You see, we specifically designed this program to process a page of text that is 60 rows by 80 columns. A page of these dimensions conveniently prints out on one 8-1/2" by 11" piece of paper. Admittedly, you should never attempt put together a 1000-page novel using this program, but for the more common job of roughing out one-page notes, *HCJWord* is the perfect choice.

CONTROL CAPSULE

HCJWord

KEY	FUNCTION	KEY	FUNCTION
ENTER	Move down a line and left	FCTN 7	Tab
		FCTN 8	Insert line
FCTN 1	Delete characters	CTRL 1	Save text
FCTN 2	Insert characters	CTRL 2	Load text
FCTN 3	Delete line	CTRL 3	Print text
FCTN 4	Roll down	CTRL 4	Erase text
FCTN 5	Next window	FCTN 9	Exit *HCJWord*
FCTN 6	Roll up		

The Edit Screen

Because the TI-99/4A's video screen cannot display a whole page of text at one time, *HCJWord* uses an edit screen that works as a window. This window shows 23 rows by 28 columns of text. By using the Roll and Next function keys (see Control Capsule), you can move this window to display any section of the page that you desire.

The topmost line of the edit screen is the status line. Most of the time, this line displays the current row and column position of the cursor within the page. At other times, the status line is used for presenting error messages and prompting the user for such information as file names and printer parameters.

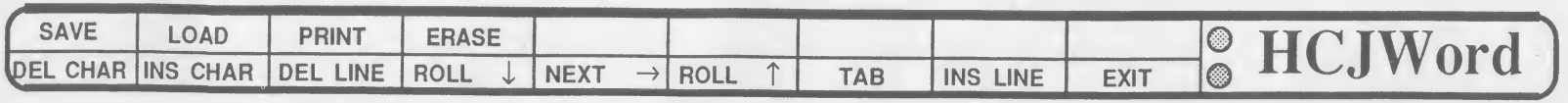
To enter text, simply begin typing. There is no "edit" mode to enter, because you are always in it. Your cursor appears as a non-blinking black square on the screen. As you will notice, the cursor moves under the text, not over it. If while entering text your cursor reaches the edge of the screen, the edit screen will automatically shift its viewing window to display the section of the page that you are typing on. The same thing happens when you try to move the cursor off the screen in any direction—the edit screen automatically shifts its viewing window accordingly. When you have reached the end of a page line, you can either press **[ENTER]** or move the cursor down to get to the next line.

Text Manipulation

HCJWord provides all the standard editing features of the TI computer, and more. You can delete characters, insert characters, erase a line, insert a line, tab right, cursor in any direction, and erase the entire page of text. Also, the cursor obligingly squashes to half its normal height whenever the editor is in insert mode—a useful form of visual feedback. All these editing features are accessed via a keypress (see Control Capsule). In Figure 1, we have provided a cut-out keyboard strip that fits above the computer's keyboard. This strip allows for ready viewing of the various keystrokes that *HCJWord* has to offer. We suggest that you make a copy of Figure 1 for use as a keyboard overlay.

Saving, loading, and printing of text is also achieved through keypresses. **[CTRL] 1** is for saving text, **[CTRL] 2** is for loading text, and **[CTRL] 3** is for printing text. Whenever you choose to save or load text, the computer prompts you for a file name. The file name must be preceded by the device (i.e., **DSK1.**), or an error will occur. Similarly, whenever you choose to print your text, the computer prompts you to enter your printer parameters. To exit *HCJWord,* press **[FCTN] 9**. If you have made any changes to your text without saving them, *HCJWord* asks if you wish to save the text before quitting.

Figure 1.



Customizing The Program

If you desire, you may modify certain aspects of *HCJWord* to suit your personal requirements. The printer parameters, for instance, always default to the following when the program is first run: **RS232.BA=9600.DA=8**. If your printer's parameters are different, simply change the string variable **PRINTER$** in line 360 to equal the correct parameters. To make the program default to **PIO**, for example, change line 360 to read:

```
360 PRINTER$="PIO"
```

Also, once **PRINTER$** has been set correctly, there's no reason to enter the printer parameters everytime you go to print your text. To stop *HCJWord* from demanding printer parameters before printing a file, delete program lines 630 and 640. With these two lines deleted, pressing **[CTRL] 3** will print your text with no questions asked.

As for saving and loading files, you can also change the default device to something other than **DSK1**. To make the program default to **DSK2**, for example, change line 370 to read:

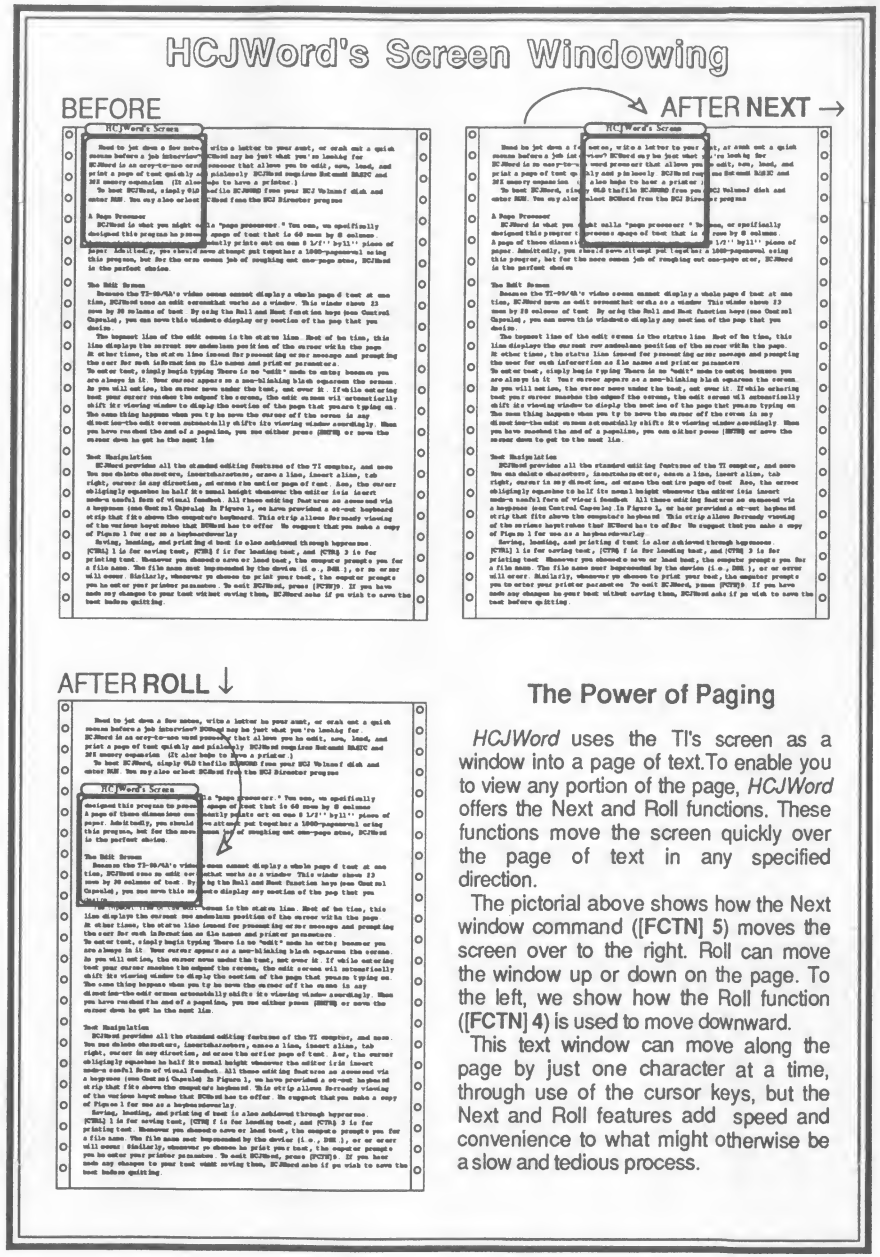
```
370 FILE$="DSK2."
```

How It Works

The engine of this program is contained in the file EDITOR_OBJ, found on your HCJ disk. This file contains the object code for a machine language text editor. This editor loads into the lower section of the 32K memory expansion unit ($2000-$3FFF). This area is also where the page of text is eventually stored. Because this area of memory expansion is unused by Extended BASIC, utilizing the editor does not limit the size of a program's BASIC code or variable space.

If you've already used the program *FormFlex*, a Feature program in this issue, you will no doubt see similarities in the *HCJWord* editor and the *FormFlex* editor. They both use the same editor kernal—EDITOR_OBJ.

Because *HCJWord* requires the file EDITOR_OBJ, you must include this file on any backup of *HCJWord* that you make, or your backup will not boot properly.



HCJournal™ DIRECTOR

About HCJ Director and CodeWorks

The directories and instructions on these two pages should help you locate any program file on your HCJ diskette and give you the necessary information to successfully RUN any program. Each system has its own specially designed HCJ Director program that allows menu-driven access to the HCJ programs in this Volume.

Each disk also contains a CodeWorks program that describes which sections of code accomplish specific tasks in our Feature programs. Full operating instructions are included in each CodeWorks program explaining how to either view the information on screen, or dump it to your system's printer.

Apple II Family

Procedures For Loading Apple II Computers

1. Place the HCJ disk in drive 1.
2. Turn on the Apple computer. The HCJ Startup menu will appear, asking if you wish to use the HCJ Director program, go to Applesoft BASIC, or format a ProDOS disk.
3. Select HCJ Director from the menu.
4. A list of the programs included on the HCJ disk will appear on the screen. Enter the number corresponding to the program that you wish to RUN and press [**RETURN**].
6. Make sure that the HCJ disk is still in drive number 1 and press [**RETURN**] again. The selected program will LOAD and RUN for you. You may press [**ESC**] before pressing [**RETURN**] for the second time, if you change your mind.

Program Name	File Name	Language
ProDOS Utilities	PRODOS*	-system file-
	BASIC.SYSTEM*	-system file-
	LINKUP*	-system file-
	FORMBUILD*	-system file-
	STARTUP*	Applesoft BASIC
HCJ Director	HCJDIR	Applesoft BASIC
CodeWorks	CODEWORK	Applesoft BASIC
FormFlex	FORMFLEX**	Applesoft BASIC
	MAKEFORM**	Applesoft BASIC
	USEFORM**	Applesoft BASIC
	COSTCOMP.FRM	-data file-
Re-Weaver	REWEAVER	Applesoft BASIC
Upsets	UPSETS***	Applesoft BASIC
Hi-Res Text	HIRESTEXT	Applesoft BASIC
	HIRESTEXT.BIN	6502 machine code
	NOTEPAD	Applesoft BASIC
	NOTEPAD.TXT	-data file-
Line Input	LIN.LOADER	Applesoft BASIC
	LIN	6502 machine code
	LIN.S	Assembler source file
	OUPUT.IT	Applesoft BASIC
	INPUT.IT	Applesoft BASIC
	GET.IT	Applesoft BASIC
	LINPUT.IT	Applesoft BASIC
WordWeave	WEAVE****	Applesoft BASIC

*ProDOS, and ProDOS Utilities, Copyright © 1983-86 Apple Computer, Inc.
**Requires either an Apple IIe with 80-column card or an Apple IIc.
***Requires a color monitor.
****This is an improved version of the Feature program *WordWeave* published in HCJ Volume 1. The program's disk access is now much faster.
Note: The Apple II Family HCJ disk contains portions of the Apple ProDOS operating system that allow the disk to be used by itself.

Atari 800, 800XL, 130XE

Procedures For Loading Atari Computers

1. Place your DOS 2.5 systems disk in drive 1.
2. Turn on the Atari computer.
3. After READY appears in the upper-left corner of the computer screen, insert the HCJ disk into drive 1.
4. Type **RUN "D:file name"** where **file name** is the file name of the program that you wish to run. Now, press [**RETURN**].
5. The selected program will LOAD and RUN for you.

***

In order to produce a disk that runs on any Atari disk drive, our programs are provided on a single-density disk. Unfortunately, due to the limited size of a single-density disk and the extensive size and complexity of this Volume's programs, we were unable to include the HCJ Director and CodeWorks programs on your HCJ Volume 2 disk—they just wouldn't fit! We plan to include these programs on all future HCJ disks.

***

Program Name	File Name	Language
FormFlex	FORMFLEX	Atari BASIC
	MAKEFORM	Atari BASIC
	USEFORM	Atari BASIC
	COSTCOMP.FRM	-data file-
Re-Weaver	REWEAVER	Atari BASIC
Upsets	UPSETS	Atari BASIC
Atari Menus	REVERSAL	Atari BASIC
	MENUS.TXT	Atari BASIC text file
Joystick-Controlled Cursor	JOYCSR	Atari BASIC
	JOYCSR.S	Assembler Source file

Note: Because some programs reconfigure your computer's memory, it is recommended that you press the [**RESET**] key or re-boot your computer if you experience problems running a new program.

Commodore 64

Procedures For Loading The C-64

1. Turn on your computer system.
2. Place the HCJ disk into the drive known as device number 8.
3. Type **LOAD "HCJDIR",8** and press [**RETURN**]. After the loading procedure is complete and the cursor returns, type **RUN** and press [**RETURN**].
4. A list of the programs included on the HCJ disk will appear on the screen. Enter the number corresponding to the program that you wish to RUN and press [**RETURN**].
5. Make sure that the HCJ disk is still in drive number 8 and press [**RETURN**] again. The selected program will LOAD and RUN for you. You may press your computer's left-arrow key before pressing [**RETURN**] for the second time, if you change your mind.

Program Name	File Name	Language
HCJ Director	HCJDIR	C-64 BASIC
CodeWorks	CODEWORK	C-64 BASIC
FormFlex	FORMFLEX	Master-64
	MAKEFORM	Master-64
	USEFORM	Master-64
	BOOT.FORM	Master-64
	MASTER64I10U-A	Master-64 run time
	MASTER64I10U-B	Master-64 run time
	COSTCOMP.FRM	-data file-
Re-Weaver	REWEAVER	C-64 BASIC
Upsets	UPSETS	C-64 BASIC
Ripples	RIPPLES	C-64 BASIC
	HRG-64	6510 machine code
Alpha Lock	ALPHALOCK	C-64 BASIC
	WWALPHALOCK	C-64 BASIC
	ALPHALOCK.S	Assembler source file
BASID Synthesizer	SYNTH*	BASID

*There was a small insect (a bug) in the original version of *BASID Synthesizer* published in HCJ Volume 1, so we are providing an exterminated version on your Volume 2 disk.
Note: Because some programs reconfigure your computer's memory, it is recommended that you restart your computer if you experience problems running a new program.

HCJournal™

Don't Miss A Single Volume —Subscribe/Renew Today... ...And Don't Forget To Tell Your Friends About Our New Look!

Home Computing Journal (HCJ) is a quarterly multi-media software subscription service containing ready-to-run productivity, education, entertainment, and utility programs on a floppy disk. The accompanying workbook contains the required support documentation plus additional technical notes and programming aids.

Artificial intelligence, database management, high-powered programming aids, realistic simulations, and specialized software for personal investing, task-specific report writing, computer-assisted design, desktop publishing, personal communications, plus entertaining math and logic excursions are just some of the projects already on our planning board.

YES, Please accept my order for

Home Computing Journal:

☐ New subscription ☐ Renewal subscription

☐ 2-Volume Mini-Subscription: { **$45** postpaid U.S. / U.S. **$52** in Canada

☐ 4-Volume Subscription { **$75** postpaid U.S. / U.S. **$89** in Canada

☐ Single-Volume Price: { **$25** postpaid U.S. / U.S. **$30** in Canada

Each quarterly Volume of HCJ includes the printed Journal plus the companion disk for your computer choice indicated below.

DISK VERSION					
APPLE	ATARI	C-64	IBM PC	IBM PCjr	TI

Please start with Volume No. ☐

Each Volume is numbered sequencially—i.e., Volume 1, Volume 2, Volume 3...

TOTAL ORDER: _____

☐ Check or Money Order Enclosed
MUST BE IN U.S. FUNDS DRAWN ON A U.S. BANK

Charge my: ☐ VISA ☐ MasterCard

Account No. ☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐

DATE EXPIRES _____ SIGNATURE _____

PLEASE PRINT ALL INFORMATION BELOW

NAME _____

ADDRESS _____

CITY _____ STATE ___ ZIP _____

TEL. NO. _____

Prices Subject To Change Without Notice.

Mail with check, money order, or credit card information to:

Home Computing Journal
P.O. Box 70248 • Eugene, OR 97401

MC/VISA orders may also be placed by telephone:

Tel. (503) 342-4013

West Coast Time (Normal Business Hours)

If you prefer not to cut this copy of your Journal, you may photocopy this form to send in with your order.

IBM PC, PCjr, and Tandy 1000

Procedures For Using The IBM PC, PCjr, Or Tandy 1000

To make use of the HCJ Director menu program on your HCJ disk you need to backup your disk. Use the following procedures to produce an autoboot backup of your HCJ disk:

If you have a dual-drive system you may start with step 1, otherwise read this paragraph first:
For those of you with a single disk drive, you may still use the commands as listed below, though you will need to pay very close attention to the prompts on the screen instructing you to swap disks from time to time. The computer will tell you to place the appropriate disk in drive B:. What it means, however, is to remove the disk from drive A: and insert the disk which would have gone in drive B:. Using a single drive may mean having to swap disks quite a few times. For those who are patient though, the rewards are worth the added work. If you have further questions consult your DOS manual on the FORMAT and COPY procedures for a single-drive system.

1. Place your DOS master disk (hereafter refered to as the DOS disk) in drive A: and turn on the power to your system.
2. Enter the command **FORMAT B: /S /V**
3. The computer will ask you to place a blank disk into drive B: to be formatted. Ensure that the blank disk is in the drive and then press [**Enter**]. After formatting, you will be asked for a volume name. Enter **HCJOURNALn** where **n** is the Volume number. Then, you will be asked if you want to format another. Respond No to this prompt which returns you back to DOS.
4. If you wish to use a color monitor enter the command **COPY A:MODE.COM B:**
5. Enter the command **COPY A:BASIC*.* B:BASIC.***
If you have an IBM compatible whose BASIC does not start with the word BASIC, then make adjustments in the command above for your version. In any case, the file on your new boot disk should always be named BASIC even if it was originally named BASICA.
6. After BASIC is copied to the new disk in drive B:, remove the DOS disk from drive A: and place the HCJ disk in drive A:
8. Enter the command **COPY A:*.* B:**
9. After the last file has been copied, remove both disks from the system. Label the new disk as **HCJ ON DISK BACKUP Volume n**, where **n** is the Volume number, and place the original disk in a safe place.
10. The new disk you have created can now be used to boot your system (start from a power off condition) and will automatically bring up a menu of programs from which you may select.

You may also use the HCJ disk without backing it up if you:
1. Start from DOS 2.1 or later.
2. If you wish to run a program with a BAT, COM, or EXE extension, simply type the file name from the DOS A> prompt.
3. If you wish to run a BASIC program, you must first enter the appropriate version of BASIC, then LOAD and RUN the program.
Note: If you have an IBM PC and the program requires a color monitor, you must enable the monitor using the appropriate DOS MODE command before running the program.

Program Name	File Name	Language
HCJ Director	HCJDIR.COM*	Turbo Pascal
	AUTOEXEC.BAT	-batch file-
CodeWorks	CODEWORK.COM*	Turbo Pascal
FormFlex	FORMFLEX.COM*	Turbo Pascal
	MAKEFORM.CHN*	Turbo Pascal
	USEFORM.CHN*	Turbo Pascal
	COSTCOMP.FRM	-data file-
	COSTCOMP.FLD	-data file-
Re-Weaver	REWEAVER.BAS**	BASICA
Upsets	UPSETS**	BASICA
PC Character Editor	CHAREDIT.EXE****	Compiled BASIC
	CHAREDIT.BAS***	BASICA
	CHARSUBS.BAS***	BASICA
	IBMFONT.FNT	-data file-
	TANDFONT.FNT	-data file-
	NUKEMAN.FNT	-data file-
Nukeman	NUKEMAN.BAS***	BASICA
GET SEG	GETSEG.BAS**	BASICA

*Program requires: DOS 2.1 or later.
**Program requires: DOS 2.1 & either Cartridge BASIC on PCjr or BASICA on PC, or GW BASIC on Tandy 1000.
***Program requires: DOS 2.1 & either Cartridge BASIC on PCjr or BASICA, Color/Graphics Monitor Adapter, and Color Monitor on PC, or GW BASIC on Tandy 1000.
****Program requires: DOS 2.1 & Color/Graphics Monitor Adapter, and Color Monitor on PC.

TI-99/4A

Procedures For Loading The TI-99/4A With Extended BASIC

1. Ensure the Peripheral Box is properly connected to the console. Turn on the Peripheral Box.
2. Place the Extended Basic module securely in the machine.
3. Turn on the TI-99/4A computer.
4. Insert the HCJ disk into drive 1.
5. Strike any key to bring up the first menu, then select Extended BASIC, and The HCJ Director program will automatically RUN.
6. Select the number of the program you wish to use, then press [**ENTER**] and the program will load and RUN automatically.

Procedures For Loading The TI-99/4A With TI BASIC

1. Ensure the Peripheral Box is properly connected to the console. Turn on the Peripheral Box.
2. Turn on your computer and insert the HCJ disk in drive 1.
3. Strike any key to bring up the first menu, then select BASIC.
4. To load the BASIC program you wish to use, type **OLD DSK1.file name** where **file name** is the file name of the program. For example, if you wish to use FormFlex type **OLD DSK1.FORMFLEX** and press [**ENTER**]. Now, type **RUN** and press [**ENTER**].

Program Name	File Name	Language
HCJ Director	LOAD	Extended BASIC
CodeWorks	CODEWORK	BASIC
FormFlex	FORMFLEX*	Extended BASIC
	MAKEFORM*	Extended BASIC
	USEFORM*	Extended BASIC
	EDITOR_OBJ	TMS 9900 assembler
	COSTCOMP_F	-data file-
Re-Weaver	REWEAVER	Extended BASIC
Upsets	UPSETS	BASIC
	UPSETSX	Extended BASIC
HCJWord	HCJWORD*	Extended BASIC
	EDITOR_OBJ	TMS 9900 object file
File Verify	VERIFY1	Extended BASIC
	VERIFY2	Extended BASIC

*Requires 32K Memory expansion.